A Framework for Addressing Communication Issues in Agile Model

Thesis submitted in partial fulfillment of the requirements for the award of degree of

Master of Engineering
in
Software Engineering

Submitted By
Parikshit Joshi
(Roll No. 801131016)

Under the supervision of:
Mr. Ashish Aggarwal
Assistant Professor

Under the co-supervision of:
Dr. Shivani Goel
Assistant Professor

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT
THAPAR UNIVERSITY
PATIALA – 147004

July 2013

CERTIFICATE

This is certified that the thesis entitled **"A Framework for Addressing Communication Issues in Agile Model"** is an authentic record of my own work carried out as requirements for the award of the degree of M.E. (Software Engineering) at Thapar University, Patiala, under the guidance of **Mr.Ashish Aggarwal** (Assistant Professor, CSED) and under the co-guidance of **Dr. Shivani Goel** (Assistant Professor, CSED) during July 2012 to June 2013.

Date: 15/7/2013

Parikshit Joshi
(Parikshit Joshi)
Roll No. 801131016

It is certified that the above statement made by the student is correct to the best of our knowledge and belief.

(Mr. Ashish Aggarwal)
Assistant Professor
CSED
Thapar University, Patiala

Shivani Goel.
(Dr. Shivani Goel)
Assistant Professor
CSED
Thapar University, Patiala

Counter signed by:

(Dr. Maninder Singh)
Head , CSED
Thapar University, Patiala

(Dr. S.K. Mohapatra)
Dean of Academic Affairs
Thapar University, Patiala

Acknowledgement

No volume of words is enough to express my gratitude towards my guides **Mr. Ashish Aggarwal** and **Dr. Shivani Goel**, Department of Computer Science & Engineering, Thapar University, Patiala, who have been very concerned and have aided for all the materials essentials for the preparation of this thesis report. They have helped me to explore this vast topic in an organized manner and provided me all the ideas on how to work towards a research-oriented venture.

I am also thankful to **Dr. Maninder Singh**, Head of Computer Science & Engineering Department and **Mr. Karun Verma**, P.G. Coordinator, for the motivation and inspiration that triggered me for the thesis work.

I would also like to thank the staff members and my colleagues who were always there at the need of hour and provided with all the help and facilities, which I required, for the completion of my thesis work.

Most importantly, I would like to thank my parents and the almighty for showing me the right direction out of the blue, to help me stay calm in the oddest of the times and keep moving even at times when there was no hope.

Parikshit Joshi

(801131016)

Abstract

There are various software development models are used in software industry. The selection of models is based on the type of software. Agile models are widely used for developing software in industries. Various types of models have been developed time to time and consistent work in the field of agile is going on. Different types of agile models have been developed as extreme programming (xp), adaptive system development, dynamic system development methodology, scrum etc. These methods provide quick response to change request, fast and frequent delivery of software products to customer and immediate feedback. Due to change in market scenario, these factors have made agile methods more popular than traditional software development methods.

The problem, in agile methods, arise when distributed team working on different module of same software. And this problem become more crucial when there is less focus on documentation. In order to overcome problem, a framework has been proposed in this thesis. Framework will provide the information that which modules are connected to current module. So if there is any change request in current module then the team leader of other module which are related to current module can be informed immediately.

Table of Contents

List of Figures

List of Tables

Chapter 1
Introduction

1.1 What is Agile Software Development

1.1.1 Agile Software Development Process

Agile software development process is a group of software development methods based on iterative and incremental development, where requirements and solutions evolve through collaboration between self-organizing, cross-functional teams[11]. Agile methods are iterative because they develop software in some iteration and they are also incremental in nature because these models focus to develop software in increments. It promotes adaptive planning, evolutionary development and delivery, a time-boxed iterative approach, and encourages rapid and flexible response to change. It is a conceptual framework that promotes foreseen interactions throughout the development cycle. The term “agile” carries with it connotations of flexibility, nimbleness, readiness for motion, activity, dexterity in motion and adjustability [38].

Agile development methods are emerging methods in software development because these encourage the rapid and flexible response to changes. Agile software engineering combines a philosophy and a set of development guidelines [3]. The philosophy promotes early delivery of software product delivery software product in small increment and less focus on documentation. The development guidelines keep focus on analysis and designing of software. Main focus of agile software development method is keeping code simple and testing often. The software process is becoming a major concern in most software development organizations as one of the ways to assure the software quality while developing software system [2]. There are various traditional development methods for developing the software. These methods are referred as heavyweight development methods because they make heavy use of documentation in various phases of development [3]. The Traditional Software Development Methods (TSDMs) are still widely used in industry because of their straightforward, methodical, and structured nature [1], as well as their predictability, stability, and high assurance [5]. Many TSDMs have been developed since the waterfall model to provide significant productivity improvements like spiral, prototype and RAD. These are not free from major problems including blown budgets, missed schedules, and flawed products [5, 6] and they have failed to provide

dramatic improvements in productivity, in reliability, and in simplicity[6]. Due to constant changes in the technology and business environments, it is a challenge for TSDMs to create a complete set of requirements up front [7]. Agile development method, such as Extreme Programming (XP), Feature Driven Development (FDD), Adaptive System Development (ASD), Dynamic System Development Methodology (DSDM) and Scrum, are developed to overcome these problems. These models are more effective than waterfall models or other traditional methods to adapt the changes or to handle the continuously changing requirements. Agile team more rapidly responses to change than TSDM teams [8]. As changes in the requirements are costly to accommodate in later phase of software the project [9], the ability to respond rapidly to change reduces project risks and their costs [10]. Although there are many positive aspects of adopting agile methodology as underlying development method, there are still some significant issues and challenges that are faced while working with agile methods.

1.1.2 History of Agile Methods

In the early 1970s, the concepts of Evolutionary Project Management (EVO) have evolved into competitive engineering [1]. A paper was presented entitled "Managing the Development of Large Software Systems," in 1970. In this paper, sequential development methods were criticized and focus was on that software must not be developed like automobile in which each element is added sequentially[13]. In 1974, a concept of adaptive software development process was introduced. Concurrently and independently the same methods were developed and deployed by the NewYork telephone company's systems development center under the direction of Dan Gielan. Agile Methods are a reaction to traditional ways of developing software and acknowledge the "need for an alternative to documentation driven, heavyweight software development processes"[13]. In the implementation of traditional methods, work begins with the elicitation and documentation of a "complete" set of requirements, followed by architectural and high-level design, development, and inspection. Beginning in the mid-1990s, some practitioners found these initial development steps frustrating and, perhaps, impossible [14].

The industry and technology move too fast, requirements "change at rates that swamp traditional methods", and customers have become increasingly unable to definitively state their needs up front while, at the same time, expecting more from their software.

As a result, several consultants have independently developed methods and practices to respond to the inevitable change they were experiencing. These Agile Methods are actually a collection of different techniques (or practices) that share the same values and basic principles. Many are, for example, based on iterative enhancement, a technique that was introduced in 1975 [16]. This group of software development methodologies was given the name ‘agile’ when a group of software development practitioners met and formed the Agile Alliance in February 2001 [27]. The agile movement could mark the emergence of a new engineering discipline that has shifted the values of the software development process from the mechanistic to the organic. Mnkandla and Dwolatzky call it Agile Software Engineering (ASE) [23]. Calling it “Agile Engineering” without the software part could confuse many as the term ‘agile’ has become a buzzword even in other disciplines that are outside software development. Agile development is a counter movement to thirty years of increasingly heavy–handed process meant to refashion computer programming into software engineering, rendering it as manageable and predictable as any other engineering discipline [12].

The Waterfall Model came first, as a way in which to assess and build for the user’s needs [23]. It began with a complete analysis of user requirements. Through months of intense interaction with users and customers, engineers would establish a definitive and exhaustive set of features, functional requirements, and non-functional requirements. This information is well-documented for the next stage, design, where engineers collaborate with others, such as database and data structure experts, to create the optimal architecture for the system. Next, programmers implement the well-documented design, and finally, the complete, perfectly designed system is tested and shipped [14].

Practitioners came to realize that methods that would respond to change as quickly as it arose were necessary [30], and that in a dynamic environment, “creativity, not voluminous written rules, is the only way to manage complex software development problems” [15]. Practitioners also began to look to other engineering disciplines for process inspiration, turning to one of the more innovate industry trends at the time, Lean Manufacturing [30]. Some believed that people inherently want to do a good job, and that managers needed to allow workers on the floor to make decisions and solve problems, build trust with suppliers, and support a “culture of continuous improvement of both process and products” [24]. Other practitioner thought that

quality was a management issue and while Japanese manufacturers were creating better and cheaper products, United States manufacturers were blaming quality issues on their workforce[24]. A project was started in the early 1990s as an attempt to unify three existing payroll systems [29] and had been declared a failure. Some developer decided to scrap all the existing code and start the project over from scratch. A little over a year later, a version of C3 was in use and paying employees. Developer were able to take a project that had been failing for years and turn it around 180 degrees. The C3 project became the first project to use Extreme Programming [26]. Similar stories echo throughout the development world. In the early 1990s, the IBM Consulting Group hired professionals to develop an object-oriented development method [26]. These professionals decided to interview IBM development teams and build a process out of best practices and lessons learned. They found that "team after successful team 'apologized' for not following a formal process, for not using high-tech tools, for 'merely' sitting close to each other and discussing while they went," while teams that had failed followed formal processes and were confused why it hadn't worked, stating "maybe they hadn't followed it well enough" [26].

1.1.3 Agile Manifesto

Experts from software fields summarized their viewpoint, saying that the agile movement is not anti-methodology[23]. In fact, experts want to restore credibility to the word methodology. Experts believe that they embrace modeling, but not in order to file some diagram in a dusty corporate repository. And they also embrace documentation, but not hundreds of pages of never-maintained and rarely used tomes [27]. Seventeen of the agile proponents came together in early 2001 to discuss the new software developments methods and emerged result was the agile software development manifesto. Representatives from Extreme Programming (XP), SCRUM, DSDM, Adaptive Software Development, Crystal, Feature-Driven Development, Pragmatic Programming, and others sympathetic gathered and the need for an alternative to documentation driven, heavyweight software development processes convened [27].

The manifesto is as follows: "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 interaction over process and tools,

• Working software over comprehensive documentation,

• Customer collaboration over contract negotiation,

• Responding to change over following a plan.

That is, while there is a value in the items on the right, we value the items on the left more"[13] . Above statement has a number of winsome aspects, not the least of which was getting 17 people to agree to it. This was a group of experienced and recognized software development experts. The word uncovering was used to assure the users that the Alliance members don't have all the answers and don't subscribe to the silver-bullet theory [22]. Second, the word *by doing it* implies that the users actually practice these methods in their own work [23]. Third, the Agile Alliance members want to help other users with agile methods, and to further improve knowledge by learning from those they try to help.

The manifesto has become an important part of Agile Movement. It defines that how agile distinguishes itself from other methods. The manifesto also characterizes the value of agile methods. The best practices of agile methods have been analyzed and compared it with traditional approaches [25]. On the point of individual and interaction over process and tools, traditional approaches relies more on process and tools than individual persons. Teams of people build software, and for that peoples in team need to work together effectively – including programmers, project managers, modelers, testers and the customers. Further, It was elaborated by Scott Ambler that five software developers and with their own tools working together in a single room can develop a better software system. The point is that the most important factors that need to consider are the people and how they work together because if the people (software development team) don't get that right the best tools and processes won't be of any use. Tools and processes are important, but they're not as important as working together effectively[28].

On working software over comprehensive documentation point, experts believe that ultimate result of building software is product. Documentation also matters but, over the years. The traditionalists made a fetish of documentation [25]. When a user is asked that whether user would want a fifty page document describing what developer intend to build or the actual software itself, user will choose working software. If that is the case, it makes more sense to work in such a manner that developer should produce software quickly and often, giving users what they prefer [28]. Furthermore, users will have a significantly easier time understanding any software that developer produce than complex technical diagrams describing its internal workings or

describing an abstraction of its usage [28]. Documentation has its place, written properly it is a valuable guide for people's understanding of how and why a system is built and how to work with the system. However, never forget that the primary goal of software development is to create software, not documents – otherwise it would be called documentation development. On the point of responding the change over following a plan, Experts believe that no side is wrong over this point. Some expert believe that [25]:

1. Customers and users do not always know what they want at the outset of a software project, and developer must be ready to accept changes during project execution and
2. Requirement change is one of the most common causes of software project failure.

One of the important point that agile methods focus is customer collaboration over contract negotiation. Some experts believe that without customer collaboration nothing is going to go well [25] .

1.1.4 Agile Principles

The 12 Agile Principles are a set of guiding concepts that support project teams in implementing agile projects. Software developer can use these concepts to implement agile methodologies in their projects. These principles are as follows[31]:-

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

11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

There are various agile methodologies. All agile methodologies follow the four values and twelve principles as outlined in the Agile Manifesto. Table 1.1 maps principles and practices to corresponding agile values.

Table 1.1 Agile Principles, Practices and Values[37]

	Value	Principle	Practice
1	Individuals and interactions over processes and tools	• Build projects around motivated individuals. • Best results emerge from self–organizing teams. • Team reflects regularly where and how to improve.	• Tacit knowledge. • Risk lists.
2	Working software over comprehensive documentation	• Measure success only through working software. • Continuous attention to technical excellence and fine design. • Simplicity is essential.	• Incremental development. • Iterative development. • Running software.
3	Customer collaboration over contract negotiation	• Business people and developers work together daily throughout the project. • Place emphasis on face–to–face communication.	• Customer collaboration. • Face–to–face meetings.
4	Responding to change over following a plan	• Keep delivery cycles short. • Satisfy customer through early and • frequent delivery. • Welcome changing	• Configuration management. • Frequent releases.

		requirements even late in the project. • Promote sustainable development pace.	

1.2 Agile Software Development Methodologies

The software process is becoming a major concern in most software development organizations as one of ways to assure the software quality while developing software system, with the software process [12]. Agile software development focuses on keeping code simple, testing often and delivering functional bits of the application as soon as they're ready. Agile methodologies get increasing attention of public in late 1990s. Each had a different combination of old ideas, new ideas, and transmuted old ideas. But they all emphasized close collaboration between the programmer team and business experts; face-to-face communication (as more efficient than written documentation); frequent delivery of new deployable business value; tight, self-organizing teams; and ways to craft the code and the team such that the inevitable requirements churn was not a crisis [2].

Agile methodologies are a lightweight, efficient, low–risk, flexible, predictable, scientific and fun way to develop software. Lightweight implies minimizing everything that has to be done in the development process, e.g. documentation, requirements etc. Efficient implies doing only that work that will deliver the desired product with as little overhead as practically possible. Low– risk implies trading on the practical lines and leaving the unknown until it is known. Predictable implies that agile methodologies are based on what practitioners do all the time. Scientific means that the methodologies are based on sound scientific principles.

Agile methodologies were originally created for small collocated teams developing software in an iterative, incremental approach, hence use of the term *lightweight methodologies* before agile was adopted [5]. However, with the use of proper modeling techniques agile methodologies can be applied to large projects [7]. The birth of agile approaches resulted in the development of several methodologies that are based on one or more of the four agile values and the agile principles as declared

in the Agile Manifesto. Agile manifesto was given on 17th February 2001, which is a formal statement which describes the agile methodologies [4].

Agile methods are a subset of iterative, interactive and evolutionary methods that are based on iterative enhancement and opportunistic development processes [5]. From a practical perspective agile methodologies, emerged from a common discovery among practitioners that their practice had slowly drifted away from the traditional heavy document and process centered development approaches to more people–centered and less document–centered approaches In all iterative products, each iteration is a self-contained, mini-project with activities that span requirements analysis, design, implementation, and test. The term interactive is used because in agile methods customer and developer interact with each other throughout development lifecycle. And customer is a part of the development team.

Presently there is work going on in an effort to fully define what agile methodologies are, hence various researchers look at agile methodologies in various ways. In this thesis the term 'Agile Methodologies' will follow a definition agreed upon by Agilists at the first eWorkshop on agile methodologies in June 2002. Agile model are described as a group of software development models that are iterative, incremental, self–organizing and emergent [40].

a) Iterative, i.e. the system is developed in some iteration. In each iteration, the problems of previous problems are solved and newer functionalities are added.

b) Incremental, i.e. the system is developed in some increments. It implies that whole system is not developed at once. The system as specified in the requirements is partitioned into small subsystems by functionality and a new functionality is added with each new release.

c) Self–organizing, i.e. the team has the autonomy to organize itself to best complete the work items.

e) Emergent, i.e. technology and requirements are "allowed" to emerge through the product development cycle. Figure1.1 sums up the definition of agile methodologies.

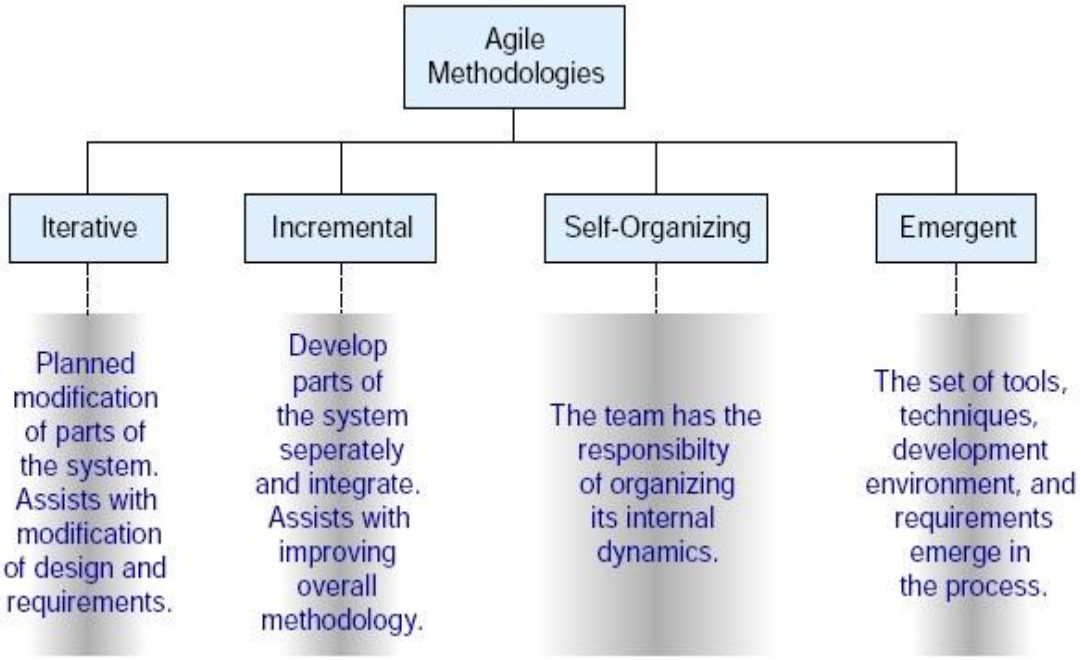

Figure 1.1 Agile Methodologies [37]

1.2.1 Agile Models

This section defines agile methodologies gives an overview of some of the most commonly used agile models in order to reveal the philosophy and fundamental approach that has been introduced by agile practices. The next few sub–sections give a brief analysis of some of the most commonly used agile models namely:

- Agile Model Driven Development
- Scrum
- Crystal
- Feature Driven Development
- Extreme Programming
- Adaptive Software Development
- ICONIX Process
- Dynamic System Development Methodology
- Lean Development

Analysis is adapted on the basis of some element adapted from different sources. These elements are as follow:-

*Techniques and tools***:** Identification of the techniques and tools used by the methodology. It should be noted that a methodology might not necessarily specify techniques and tools.

Methodology outputs: Deliverables to be expected from each specified phase of the methodology. This usually provides some guidance that analysts can use to evaluate their progress.

Methodology product: This element describes what somebody gets if they buy the methodology. This usually ranges from software to documents such manuals, books, online access to all kinds help and training. The product normally changes because of the changes in technology and available tools.

Roles and responsibilities: These refer to allocation of specific roles through which the software production in a development team is carried out. This element essentially outlines who does what in the project team [38].

Adoption and experience: This is includes a clear description of the origins of the model whether it was developed from practice –commercial background or from theory –academic background. Another point about adoption and experience is an indication of the number of organizations that are using the methodology i.e. user base, very difficult to determine. The types of users of the methodology would also fall under this element, i.e. is it used by developers or management? Associated with this notion are the skill levels that are required to use the methodology [38].

*Methodology scope***:** The level of detail to which the methodology's development framework is spelt out. Of particular importance are the limitations of the methodology[37.] The scope is defined in the life cycle of the methodology.

1.2.1.1 Adaptive Software Development:-

The structure of adaptive software development (ASD) is illustrated below in figure 1.2 The Adaptive Software Development methodology was developed by Jim Highsmith as a technique for building complex software and system. In ASD, collaboration is as much source of order in our complex interaction as discipline and engineering. There are three main phases:- speculation, collaboration and learning. During speculation, project is initiated and planning is conducted. In this phase, customer's mission statement, project constraints and basic requirements are gathered. During collaboration phase, people work together. But collaboration is not easy.

People working together must trust one other to criticize without animosity, assist without resentment and work as hard as they do.

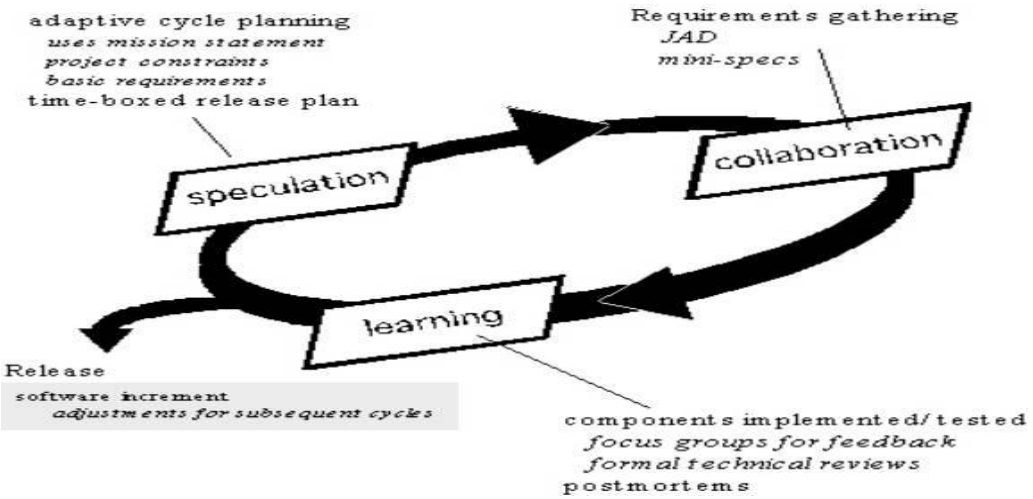

Figure 1.2: Adaptive Software Development[37]

During learning phase, ASD teams begin to develop the software component. ASD teams learns in three way:-focus group, formal technical review and postmortems. Philosophy:- The ASD philosophy is based on five primary goals of software management as stated by Highsmith [15], *adaptive culture*, which presents organizations as complex adaptive systems, and creates emergent order out of a team of individuals. The second goal is *adaptive frameworks*, which provides a series of frameworks to help an organization employ adaptive principles. Next, the third goal is *adaptive collaboration*, which setup collaboration, i.e., the interaction of people with similar and dissimilar interests to jointly innovate, which becomes the organizational basis for generating emergent solutions to product development problems. Collaboration is addressed according to interpersonal, cultural, and structural relationships [37]. The fourth goal is *adaptive scale*, which provides a path for organizations to adapt the approach on larger projects. The fifth goal is *adaptive management*, which replaces the culture of command–control management with a flexible management style that provides for wider participation in decision–making, and empowerment. This in fact means that "leadership" replaces "command" and "collaboration" replaces "control." [37]

ASD applies the old technique of Joint Application Development (JAD) sessions. Planning is based on results, it is also known as component–based planning, which plans based on a group of features (or deliverables). Spreadsheets are used for

component–based planning [35]. Customer focus group review, is another technique that is very useful for planning. Another technique is time–boxing which is the setting of fixed delivery times for iterative cycles and projects; this technique is in fact about focusing and re–evaluating the project's mission [35].

ASD is essentially a management philosophy for agile projects and therefore limited to project management activities. The limits of this methodology when it comes to team sizes depends on the size of the project, but like any other agile methodology the level of agility decreases as the team gets larger. The methodologies of communication determine the rigor in a project and the support for distributed development [35].

1.2.1.2 Crystal Model

Overview of crystal methods shown in following figure 1.3

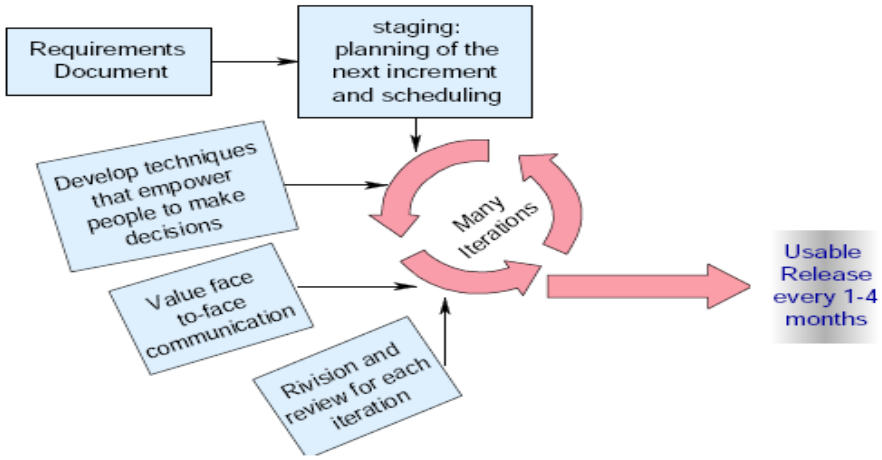

Figure 1.3: Agile Crystal Methodology[37]

Crystal Orange's process can be illustrated by the activities of one increment, which are starting from a requirements document [35], Next comes staging which is the planning of the next increment and scheduling. The next phase is the revision and review of each of the several iterations (each iteration has the following activities: construction, demonstration and review) in an increment. The three factors that are central to the Crystal process are according to Highsmith [35].

a) Communication load –The idea is that as the number of people in the project team increases the easy of communication is compromised.

b) System criticality –As system criticality increases, the size of the team also increases and the communication becomes less clear which defines another colour in the Crystal family of methodologies.

c) Project priorities –Here the emphasis is on the way of working, which is fashioned by the priorities of a project such as; time to market, cost reduction, exploration, or legal liability.

Philosophy:- The crystal philosophy can be summed up in three fundamental principles; the first is a human–powered and adaptive principle, which means that the project success is based on empowering the people involved to make decisions and enhance their work, i.e. think about people, their skills, their communications and their interactions [34]. The second is ultra–light, which means that for whatever project size and priorities, documentation and any paperwork must be kept minimal, in other words keep deliverables, and reviews to a very small number, and writing standards must be very light. The third principle is the "stretch –to–fit"[33], the principle being that it is better to start with something too light and add as needed (rather than starting with something too heavy and trying to remove) [34].

1.2.1.3 Agile Scrum Model

Structure of scrum is illustrated in figure 1.4. Scrum is an agile method for completing the complex project. Scrum is an activity used in rugby match, as the whole process is performed by one cross-functional team across multiple overlapping phases, where the team "tries to go the distance as a unit, passing the ball back and forth"[9]. The scrum model was initially developed by Jeff Sutherland and his team during early 1990s. Scrum originally was formalized for software development projects, but works well for any complex, innovative scope of work. Today there are records of Scrum used to produce financial products, Internet products, and medical products. Scrum emphasizes the use of a set of software process patterns that have been proven effective for project with tight timelines, changing requirements, and business criticality [3]. The Scrum philosophy is based on the empirical model of industrial process control used to control complex and unpredictable systems[32]. The fundamental notion of Scrum is that the set of variables that constitute software development such as people, requirements, resources, technology etc are not predictable and hence the use of rules for a defined process in software development would not be appropriated. This scenario puts software development under complex

unpredictable systems and the Scrum methodology then spells out the best practices that can be used to manage such processes.

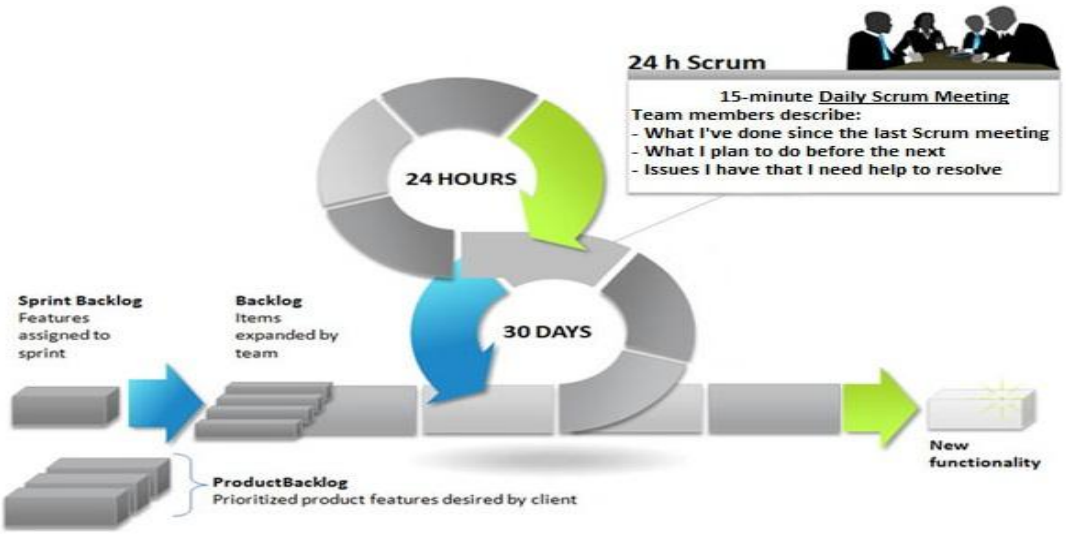

Figure 1.4: Scrum Model[9]

In scrum framework lifecycle activities include creation of backlogs, followed by sprints, scrum meeting and demos to customer. Initially a customer or product owner creates the backlogs/ requirements and priorities' the backlogs. Items can be added to backlog at any time that added the value to the customer. Project manager can access the backlogs and can update priority at any time. When the backlog is completed then sprint phase is started. In this phase, work units, that are required to achieve a requirement defined in backlog is defined i.e. sprint consists of work unit that defines that how to implement specified backlog. When the sprints are defined then, sprint teams have some time to implement these sprints usually two to four weeks. Scrum master is the key personnel in the project which keeps the team focused on their goal. Scrum meetings are daily short-meeting of scrum teams with their scrum master. During the scrum meeting several question are asked by the scrum master to their scrum teams such as what they have did since last meeting, what they will do by the next meeting. This scrum meeting helps to uncover the potential problem. When an increment of the software is developed or any backlog is created/implemented than it is demonstrated to the customer. This phase is known as Demo. In this phase customer also evaluates the functionality of developed increment. If the customer is satisfied with the functionality of current software increment than scrum team begin

to work on another backlog otherwise changes required by customer are logged and cycle starts again.

Teams should be small comprising about seven to ten people. It can be scaled to larger numbers. The methodology is aimed at project management in changing environments[38]. Planning game is a useful technique that results in a list of prioritized requirements called a Backlog list. Sprint is a technique where the prioritized list of requirements are developed into an executable product without allowing further changes to requirements within the thirty–day iteration [32]. Daily Scrum is a great project monitoring technique that gives the management control of the project as they know what happens on a daily basis and have to solve the impediments. There are no specific tools mentioned but most project management tools would be useful. Scrum Master ensures Scrum practices and values are followed up to the end of the project. Product Owner manages the project and controls and makes visible the Product Backlog list. Scrum Team has authority to decide actions and is self–organizing so as to complete a Sprint. Customer participates in product backlog items [32]. Management makes final decision and participates in setting of goals and requirements.

1.2.1.4 Agile Model Driven Development approach

Structural model of AMD approach is illustrated in following figure 1.5.

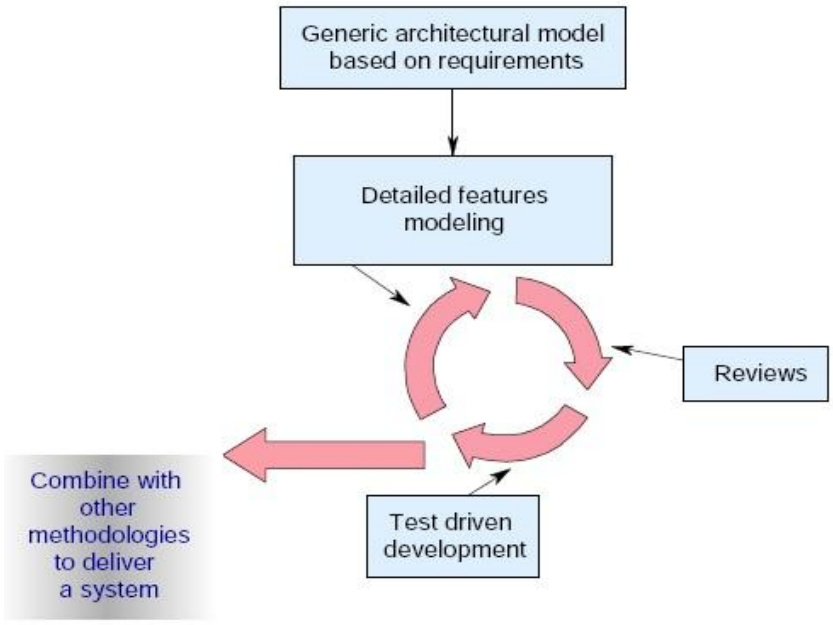

Figure 1.5: Agile Model Driven Development Approach[37]

AMDD is the agile version of Model Driven Approach. Model-driven development means that code and tests are derived from (UML) models. AMDD drops the requirement that models be formal and complete. Instead, models only need to be good enough to reach the real goal: the next release. AMDD often eschews the use of CASE tools in favor of a digital photograph of a diagram drawn on a white board. AMDD is not a complete methodology but a modeling framework, hence it does not prescribe a process but works well with other agile methodologies [37].

No specific team size is mentioned but the generally, methodology aims for small teams. The AMDD framework can be combined with all non–modeling agile methodologies [25]. Focus on working software is the main goal of this methodology. The underlying principle is to keep the amount of models and documentation minimal but advanced enough to support meticulous design [41]. The principle in AMDD is that whatever has to be done to model the problem space or the solution must be as minimal as possible without compromising value as perceived by the stakeholders. Simplicity and communication are virtues in AMDD. Agile Model Driven Development provides a modeling framework for scaling up of agile methodologies to deal with large projects.

1.2.1.5 Feature Driven Development

The structural overview of model driven development is illustrated in figure 1.6.

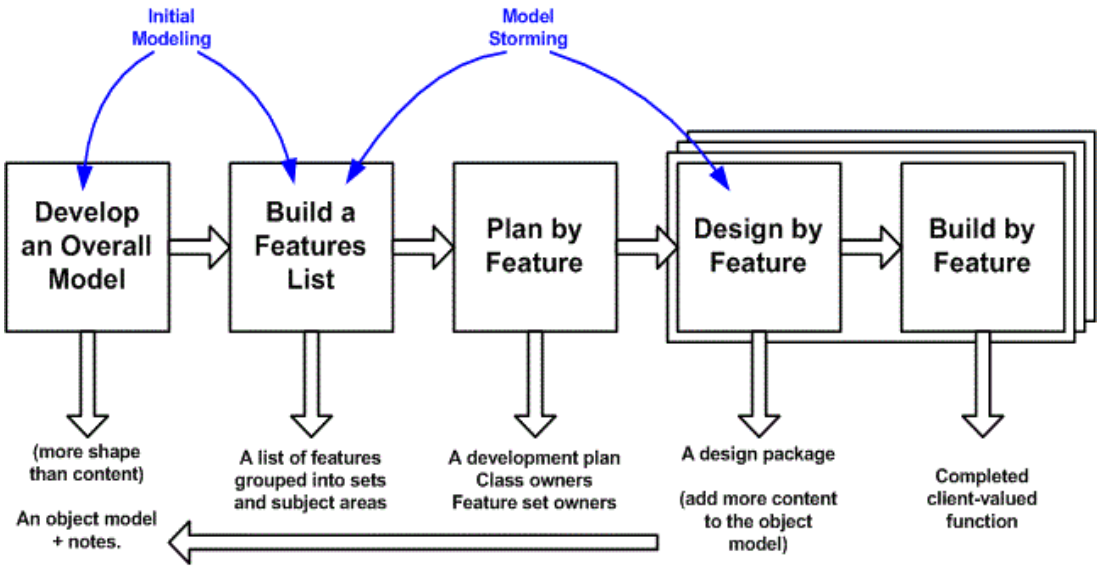

Figure 1.6: Feature Driven Development Model[42]

Feature-driven development (FDD) is an iterative and incremental software development process. It is one of agile method for developing software and forms part of the Agile Alliance. FDD blends a number of industry-recognized best practices into a cohesive whole. On Object Oriented (OO) modeling of a project to represent it in the simplest possible model that graphically captures the realities of the project. The immediate benefit of this is apparent to the programmer in the form of the short iterative feature–based planning and design process. The manager also benefits from the vivid picture of the project progress and status, which makes steering of the project easier [43]. FDD only deals with design and implementation and has to be combined with other agile processes. These practices are all driven from a client-valued functionality (feature) perspective. Its main purpose is to deliver tangible, working software repeatedly in a timely manner. FDD was first introduced to the world in 1999 via the book Java Modeling In Color with UML, a combination of the software process followed by Jeff DeLuca's company and Peter Coad's concept of features [42]. In the context of FDD, feature is a client valued function that can be implemented in two weeks or less.

Features are expressed in form <action> <result> <object> [42]. For example, "Calculate the total of a sale". Features are as important in FDD as user stories are in Extreme Programming. Features provide a better view of system as user can describe them more easily and can define how they are related to each other. Since feature in a deliverable, team deliver feature in every two weeks and project planning, scheduling and tracking is done with the help of feature rather than software engineering tasks. It's development framework includes five collaborative activities, first two activities establishes overall shape and last three activities are iterated for each feature. Process starts by developing an overall model, which provide high level scope of the system and its context. Next activity is build feature list, in this activity the knowledge that was collected in first phase/activity is used to identify the feature list. For this, domain area is functionally decomposed into subject area and subject area contains business activity and business activity forms feature. Thus the feature are client valued function represented in form of <action><result><object>.

When the feature list is build, next step is to develop a plan by feature. In this step, each feature is organized as classes and assigned to the chief programmer. In design by feature phase, chief programmer selects the set of feature that is to be delivered in two weeks and make a detailed sequence diagram together with the other class owner

and refines the overall model. And finally the design inspection is held. After a successful design build by feature step take place. This is the main implementation step. The class owner or chief programmer develop the actual code for the feature and after the unit test and code inspection the completed feature is added to main build. If the deadline pressure is high then it is difficult to determine if feature are properly scheduled [10]. To mark the progress of each feature, FDD defines several milestones that must be completed sequentially for each feature. These are domain walkthrough, design, design inspection, code, code inspection, promote to build.

1.2.1.6 Extreme Programming

The graphical overview of Extreme programming is shown in figure1.7

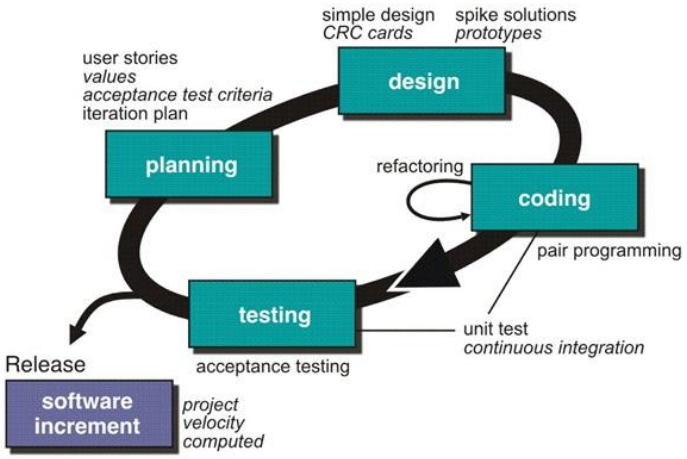

Figure 1.7: Agile Extreme Programming[10]

Extreme Programming (XP) is one of popular Agile Methodology. It has already been proven to be very successful at many companies of all different sizes and industries worldwide. It advocates frequent "releases" in short development cycles (timeboxing), which is intended to improve productivity and introduce checkpoints where new customer requirements can be adopted. Phases of XP includes planning, designing, coding, testing and then release the software increment. In first phase, initially, customer requirements are collected. These requirements are referred as user stories. These user-stories are analyzed by project manager, team members and the value is given to every user story. Then these user stories are ranked or prioritized according to their value. The acceptance criteria is set and interaction plan is defined. Only the project manager has right to change the priorities of user stories. Now first phase is

over here and second phase is started. In designing phase, CRC are generated and spike solution of the problem is given. CRC cards stand for class responsibility collaborator cards, which is used to identify and organize the object oriented classes relevant to the current software increment. If difficult design problem encountered as a part of design then Extreme Programming recommends immediate creation of operational prototype of that portion known as spike solution. And if it is the second phase of iteration, then refactoring is done which is process of changing software in such a way that it does not alter the external behavior of the system yet improves the internal structure. In third phase of Extreme Programming, which is coding, user stories are developed and the team should not move to code, but rather then develop a series of unit test which will be used to exercise each of user stories in the current release[3]. Once the unit tests have been created the developer is better able to focus on what must be implemented to pass the unit test. Key concept is to use pair programming in which two programmer work together on a one computer work station as it provides the mechanism for real time problem solving and real tile quality checking. And the last phase is testing. Extreme Programming encourages the regression testing whenever code Is modified.

XP is based on four pillars from which emanate the XP practices, i.e.: technical mastery of programming, aesthetics of code, co–dependency between business and programmers, and building of teams at startup. From these fundamental concepts some important principles are derived such as; *communication* where XP values mutual exchange of knowledge about the problem environment as key to the success of the project, and *simplicity* says the team must build the simplest thing that can work, *feedback* means that the team accepts changes the customer proposes after reviewing the delivered iteration[3].

Pair programming is a fundamental technique that has contributed to the popularity of XP. Test–first programming means that the system is tested as it is built instead of waiting until the end to test. Refactoring is a technique that allows for change in the design. The other techniques are on–site customer and collective ownership of code. XP does not specify tools for programming. The decision on the tools is determined according to the project's needs.

Programmer writes tests, and code. Customer writes stories, functional tests and sets implementation priority. Tester helps customer write functional tests and runs functional test and maintains testing tools. Tracker gives feedback on accuracy of

estimates and traces progress of iterations[43]. Coach guides team to follow XP process. A consultant is an external member who guides the team to solve problems. Manager makes decisions.

1.2.1.7 ICONIX Process

The overview of ICONIX is illustrated in figure 1.8 .

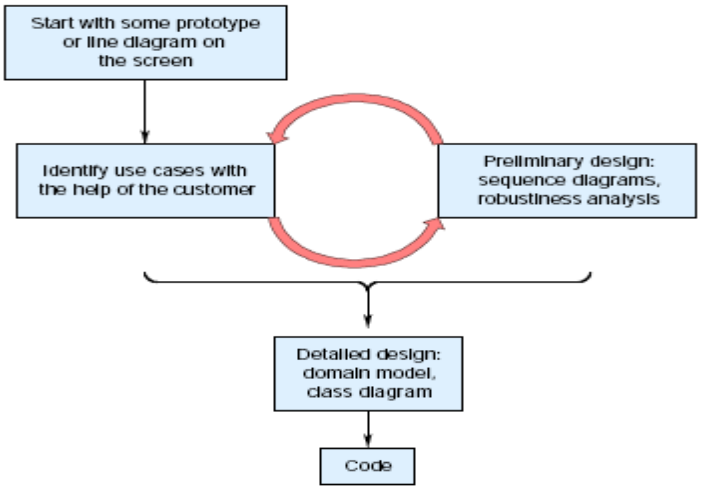

Figure 1.8 Agile ICONIX Model[44]

The ICONIX process starts from prototypes or line diagrams of the screens, after user assurance of the prototype, followed by identification of use cases on use case diagrams, then comes the writing of use case text, next comes the refinement of text in robustness analysis, which leads to detailed design on sequence diagrams, followed by breakdown of the system along use case boundaries to derive a detailed object model [44]. *ICONIX* is a software development methodology which predates both the Rational Unified Process (RUP), Extreme Programming (XP) and Agile software development. Like RUP, the ICONIX process is UML Use Case driven but more lightweight than RUP. Unlike the XP and Agile approaches, ICONIX provides sufficient requirement and design documentation, but without analysis paralysis. The ICONIX Process uses only four UML based diagrams in a four step process that turns use case text into working code.

A principal distinction of ICONIX is its use of robustness analysis, a method for bridging the gap between analysis and design. Robustness analysis reduces the ambiguity in use case descriptions, by ensuring that they are written in the context of

an accompanying domain model. This process makes the use cases much easier to design, test and estimate

The ICONIX Unified Modeling approach's philosophy was derived from a combination of the most commonly used Object Oriented modeling techniques; Booch, Object Modeling Technique (OMT) and Object Oriented Software Engineering (OOSE). It centers on building fine object models through streamlining UML diagrams to a minimal but sufficient set. Techniques are then applied to provide a quick, efficient, iterative and incremental move from use cases to code. In fact the ICONIX philosophy is based on closing the gap between use cases and sequence diagrams (detailed design) [44].

1.2.1.8 Dynamic System Development Methodology

The overview of Dynamic System Development Methodology (DSDM) is illustrated in following figure 1.9

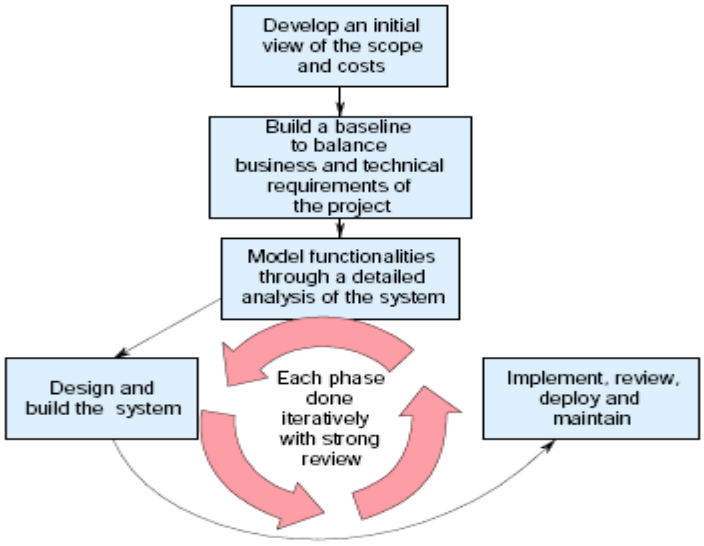

Figure 1.9: Dynamic System Development Methodology[10]

DSDM is an agile software development approach that provides a framework for building and maintaining system which meets the time constraints through use of incremental prototyping[10]. DSDM is an iterative development approach like other agile development process. When we are using DSDM then identification of crosscutting requirements is necessary throughout the DSDM development lifecycle.

The DSDM consortium is a group of member companies. The consortium has defined the life cycle activities of DSDM. The following is the brief discussion of DSDM

lifecycle activities. Process start with feasibility study, followed by business study, functional model iteration, design and build iteration and finally implementation phase. This is an iterative model i.e. if customer is not satisfied with current increment then process starts again. During the feasibility study, high level of separation of concern can take place. In this phase, high level functional or business requirement are defined and then asses that whether the application is viable candidate for DSDM process. During the business study phase of the DSDM, non-functional requirements are identified such as maintainability and other quality attribute. Next phase is functional model iteration.

The prime focus of functional model iteration is on prototyping to elicit requirements [7]. In this phase a set of prototype is produced that demonstrates the functionality for the customer. During design and build iteration, the focus is on ensuring that the prototypes are sufficiently well engineered for operational use and the functional prototypes are refined to meet the non-functional requirements. This phase ensures that each prototype has been engineered in a manner that will enable to provide operational business value to the end user. Now the last phase comes which is implementation phase. In this phase the software increments are produced or developed on the basis of prototype and then it places the latest software increment into the operational environment. It should be noted that changes may be requested by customer after the completion of increment. In this case DSDM work will continue by returning to the functional model iteration activity. DSDM philosophy can be summed up as follows:-

- Development is a team effort, which combines the customers' knowledge of business requirements with the technical skills of IT professionals.
- High quality demands fitness for purpose and technical robustness.
- Development can be incremental, i.e., not everything has to be delivered at once, and delivering something earlier is often more valuable than delivering everything later.
- The law of diminishing returns applies, which means that resources must be spent developing features of most value to the business.

1.2.1.9 Lean Software Development

The overview of Lean Software Development (LSD) is illustrated in following figure 1.10

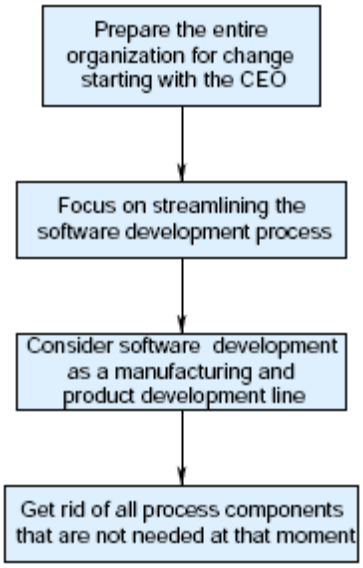

Figure 1.10: Lean software Development[45]

LSD philosophy is based on the adoption of a radical change tolerant approach to risk leadership in software development. The fundamental principle is to create change tolerant software with one–third the human effort, one–third the development hours, one–third the investment in tools and methods and one–third the effort to adapt to a new market environment [45,46]. For most organizations this scenario amounts to a turnaround on management policies hence in order for LD to be accepted it aims at changing the attitude of CEOs about change in an organization. LD is based on the Lean Thinking concept from Lean Production, that let customers delay their decisions about what exactly they want for as long as possible [37], and when they ask for something it must be given to them so fast they don't have time to change their minds.

1.2.2 Common Properties of Agile Methods

Agile is a different way of managing projects and development teams. A small group of people got together in 2001 to discuss their feelings that the traditional approach to managing software development projects was failing far too often, and there had to be a better way. They came up with the agile manifesto, which describes 4 important values that are as relevant today as they were then. It says, they 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 [10].

Ever since then, the use of methods that support these values has become increasingly popular. From the detailed study of four agile methods a brief discussion of common properties in all agile method is given as:-

- Agile methods applies Pareto principle rule that is also known as 80/20 rule.
- Focus on frequent delivery of the product.
- Testing is integrated throughout the lifecycle.
- Complete each feature before moving to next.
- Changes in the requirements are welcomed even late in the development cycle.

1.2.3 Comparison of some Agile Methods

In this section, four agile method discussed above are compared with each other based on various factors. Comparison is listed below in tabe-1.2 based on detailed study of four agile methods.

Table 1.2: Comparatively Analysis of Some Agile Methods

Attribute	Extreme Programming	Dynamic System Development Methodology	Feature Driven Development	Scrum
Proposed by	Kent Beck in 1999	DSDM Consortium in 1994	Jeff De Luca in 1997	Sutherland and Schwaber in 1995
Philosophy	Programmer Centered	objective of "jointly developing and promoting an independent RAD	Use feature list to manage functional requirements and development	whole process is performed by one cross-functional team across multiple

		framework" by combining best practice experiences	tasks	overlapping phases, where the team "tries to go the distance as a unit, passing the ball back and forth"[9]
Focus on	User Stories and their value and acceptance criteria	Time constraints and prototypes	Feature which is a client valued function	Backlogs and sprints
Prototype	Recommend use of operational prototype	Builds a set of prototype in functional model iteration	Does not recommends creation of prototype	Recommends creation of operational software increment instead of prototype
Addresses risk	Early in lifecycle	Early in lifecycle	In middle of lifecycle	Early in lifecycle
Supporting tools	Ex:- xm.tracker-1.0.6	DSDMAtern	FDD Tools 2.0, Snap-on Tools	Scrumwise, Eylean
Iterative process	Yes	Yes	Yes	Yes
Management activity completed in	Planning	Business Study Phase	Plan by feature	Depends on scrum-master

phase				
Artifacts	Working software , code, unit test cases, user manual and installation guide	Working software , documents to explain model and enable maintenance, user manual and installation guide	Working product, document consisting of feature, user manual and installation guide	Working software, user manual and installation guide
Team Member	Chief Programmer, programmer, tracker, consultant, customer	Executive sponsor, Ambassador user, project manager, Technical coordinator, Team leader, Solution Developer	Project Manager, chief architect, Development Manager, chief Programmer, Class owner, Domain Expert	Product owner, Scrum master, Scrum teams
Phases	Planning, Design, Coding, Test	Feasibility study, Business Study, Functional model iteration, Design and build iteration. Implementatio	Develop an overall model, Build a feature list, Plan by feature, Design by feature Develop by feature	Product Backlog, Sprints, Scrum Meeting, Demos

		n		

1.2.4 Traditional Software Development Methods Vs Agile Software Development Methods

The Traditional Software Development Methods (TSDMs) are still widely used in industry because of their straightforward, methodical, and structured nature [48], as well as their predictability, stability, and high assurance [50]. Though many TSDMs have been developed since the waterfall model to provide significant productivity improvements, they are not free from major problems including blown budgets, missed schedules, and flawed products [50, 55], and they have failed to provide dramatic improvements in productivity, in reliability, and in simplicity [55]. Due to constant changes in the technology and business environments, it is a challenge for TSDMs to create a complete set of requirements up front [26]. Agile development method, such as Extreme Programming (XP), Feature Driven Development (FDD), Adaptive System Development (ASD), Dynamic System Development Methodology (DSDM) and Scrum, are developed to overcome these problems. These models are more effective than waterfall models or other traditional methods to adapt the changes or to handle the continuously changing requirements. Agile team more rapidly responses to change than TSDM teams [42]. As changes are costly to accommodate later in the project [8], the ability to respond rapidly to change reduces project risks and their costs [9].

Table 1.3: Difference between AM and TSDM

Attribute	Agile Methods	TSDMs
Customer Participation	Customer is part of development team	Not actively participates
Team Empowered to make decision	Yes	No
Increment Size	Small	Depends on project
Primary Focus	Rapid delivery of software product to customer	High Assurance

Testing	Integrated throughout the lifecycle	Separate phase is used for testing
Communication between team member	Communication is high	Communication is low
Quality Process	Integrated in lifecycle	Separate and well defined Quality process
Lifecycle phases	Iterative and incremental	Depends on model used
Requirement	Changes throughout the lifecycle of product	Are fixed
Documentation	Not too much important	Very important
Team size	Small teams	Depends on project
Refactoring	Inexpensive	Expensive

1.3 Communication Problems

Agile development method, such as Extreme Programming (XP), Feature Driven Development (FDD), Adaptive System Development (ASD), Dynamic System Development Methodology (DSDM) and Scrum, are developed to overcome problems of Traditional Software Development Methods (TSDMs). These models are more effective than waterfall models or other traditional methods to adapt the changes or to handle the continuously changing requirements. Agile team more rapidly responses to change than TSDM teams [7]. As changes are costly to accommodate later in the project [8], the ability to respond rapidly to change lowers project risks and their costs[9]. Although there are many positive aspects of adopting agile methodology as underlying development method, there are still some significant issues and challenges that are faced while working with agile methods. Communication is one of the main challenge in agile methods.

Agile teams require more communication to other agile teams and customer as compares to TSDMs teams because in agile software projects, requirements are

continuously changing in nature and agile teams welcome changes even late in development lifecycle. In case of distributed teams, it is difficult for agile teams to maintain communication. Large organizations usually distribute teams across several physical locations. Various organizations found that agile teams have difficulties interfacing with other teams. Such as Nokia and Motorola found that, agile methodology has difficulties in maintaining communication between distributed team, while using XP for their projects [49].

In large organizations, where distributed teams are working on different subsystem/module of a single project, effective communication is difficult to achieve (because such development teams do not use proper documentation as they are agile and various decisions are taken by team not by management).

And this problem become worst when there is coupling between two modules as team member don't know that which subsystem will be affected by change suggested by customer or by development team.

1.4 Chapter Summary

Further chapters of thesis are organized as below:

In chapter 2, a survey has been conduct. This includes different communication issues related to agile models and factors affecting the communication in agile models. This part also provides the idea about different types of communication in agile models and technology adaptation lifecycle.

Chapter 3 concentrates on problem statement i.e. what is the actual problem and how it can be solved. Here objectives of thesis are identified.

In chapter 4, the solution of the problem defined in chapter 3 is given. It includes several steps for converting DFD into PERT chart and then PERT to dependency list. This chapter also includes the working of framework.

Chapter 5 includes two case study to show the working of the framework. Case study of order processing system and admission management system has been included.

Chapter 6 Concludes and defines the future scope of the work. This chapter also describes the contribution of thesis.

Chapter 2

Literature Survey: Communication Issues In Agile

There are many positive aspects of agile development methodology, such as less dependent on documentation, welcome changes in software even late in development lifecycle, lower risk of development [8, 10, 51, 53]. There are also various issues and challenges arise while adopting and working agile software development methods. These challenges are no fixed prototype of documentation, lack of communication, customer involvement, serial thinking, closed mindedness, office politics, black and white mind-set, fear of change, specialized skills, outdated skills, documentation-heavy mind-set, and do-it-all-at-once attitude [7, 53, 54].

Organizations must carefully evolve toward the best balance of agile and plan-driven methods that fits their situation [48]. When we migrate from TSDMs to agile based development then this migration requires the adaptation of new technology and tools. This adaptation process is characterized by Moore's technology adaptation lifecycle (TALC) as shown in figure 2.1.

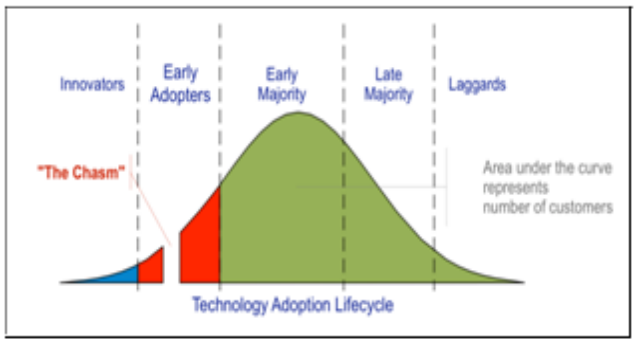

Figure 2.1: Technology adaptation lifecycle by Moore[57]

Technology adaptation curve is bell shaped or normal distribution curve over time. It can be seen that there are five phases in curve that every new technology must pass during it's adaptation to organization. These phases are innovators, early adapters, early majority, late majority and laggards. Innovators are people who like technology for technology's sake. These people are technology enthusiasts. Positive response of the innovators is necessary for adaptation of new technology. Next is early adaptors, these are visionaries because it is easy for them to understand and imagine the benefit of new technology and they will take a risk for trying new technology for competitive

advantages. Next is early majority, these people are pragmatists. These people are ready for buy the new technology when someone other has tried and worked out the bug. Late majority are conservatives who are less educated, fairly conservative and prefer traditional technology over new technology. Laggards are highly traditionalist who adopt technology very late or may not adapt new technology. These reject new and innovative technology.

There is a gap between early adopters and early majority known as technology chasm. Agile processes are currently in early adopters phase [49]. These condition occur when there are some people in the organization who are ready to accept new technology with some risk involved in it (early adopters) and some people want new technology only when it is well tested and proved effective. Common challenges while adopting agile process are serial thinking, closed mindedness and fear of change [58]. Serial thinking is related to people who are working in IT/Software industries from many years. They have a serial approach of developing a software in their mind, that is, analyze, design, coding, testing and maintenance. Such people should be given the appropriate environment to learn basic principles and working of agile methods. There are various challenges faced when we migrate from traditional development methods to agile development methods because changes in development methods cannot be accomplished merely by replacing current tool and technology with newer tools and technology. There are various issues/challenges on which we must pay attention. Such challenges/issues may affect the culture, behavior and structure of the organization [59]. And in terms of team, migration from TSDMs to agile requires more powerful teams which can make decisions and such condition may affect moral of non-agile developers. And there are also some technological changes. Another problem in migration from TSDMs to agile is the change in underlying process of the organization because organization has to shift from a plan-based approach to such an approach which considers that everything is uncertain (agile).

In large organizations, the problem arises when we introduce agile method as a new development methodology because in large organization there is already a well defined and well tested traditional approach exists. We just cannot replace the older development approach with newer ones because this will take some time and business will be affected. So integration of the newer process is done and problems faced are fear of change to some people who are working from a long time, so specialized skills are required. If the organization is big one then it is a high probability that there will

be distributed teams working on the same project from different locations and agile teams have difficulties in interfacing with each other [58]. So if an organization introduces agile as new development methodology then it may face problems and it has to modify some principles of agile and that may take time. Sometimes there may be clash when we are integrating agile methodology with older one such as clash in the principle of agile and existing methodology and that may result in double work. For example, while developing a software agile principle, consider that unit testing is integrated throughout the development lifecycle of the software. While traditional methodologies such as, waterfall, has separate phase for testing. In such condition double work may be done.

In agile methodology, refactoring is an important concept and often used when we have to improve the internal structure of that module while does not affect its external structure. In case of distributed teams in large organization, it creates a problem when one team refractors the module developed by other team and it may degrade the quality of that module [54].

Other challenges of agile methods include deficiency in producing generalized software, limited support for building reasonable artifacts, limited support for safety critical software [58], less focus on documentation, communication, customer involvement, de-emphasis on upfront design work, working environment, lack of upper management's commitment, need for accurate estimates for invoicing/billing processes, amount of rework is higher as architectural issues are not discussed, loss of privacy. Although less focus on documentation improves productivity as a lot of time is saved that is spend on creating and maintaining documents, but it creates the problem at the time of maintenance of the software or when one of the key developer leaves the organization. Upper management's commitment is very necessary when organization is moving towards agile. It is seen that management is ready for development through agile principles but hesitates in giving the decision making power to teams and that creates difficulties for agile teams. Customer involvement is a key factor for the success of agile methods. Customer is a part of development team but it is seen that after two or three month customer doesn't involve in teams as much as necessary neither customer takes responsibility of project being late due to changing requirements. Communication is one of the key challenge in agile methodology [60]. Agile teams require more communication to other agile teams and customer as compares to TSDMs teams because in agile software projects,

requirements are continuously changing in nature and agile teams welcome changes even late in development lifecycle. In case of distributed teams, it is difficult for agile teams to maintain communication. There are various issues and modes related to communication in agile methods. We will explore them in later sections. In table 2.1, we have listed down the challenges with serial number.

Table 2.1: Challenges and Issues

#	**CHALLENGES AND ISSUES**
1	lack of communication
2	customer involvement
3	serial thinking
4	closed mindedness
5	office politics
6	black and white mind-set
7	fear of change
8	specialized skills required
9	Use of outdated skills in agile development
10	do-it-all-at-once attitude
11	distributed teams
12	Danger of clash in principles when we are integrating agile with older technology
13	deficiency in producing generalized software
14	limited support for building reasonable artifacts
15	limited support for safety critical software
16	less focus on documentation
17	de-emphasis on upfront design work
18	lack of upper management's commitment

2.1 Communication issues in Agile Methods

Communication in agile modeling, is transferring the information from one person to another. In agile modeling, communication plays an important role in the success or failure of agile projects or we can say that a good and effective communication leads

agile project to success. A lot of rework is done due to bad communication. When we think about communication then first question in our mind is that how do we communicate. Alistair Cockburn has described various modes of communication based on media richness theory [61]. Figure 2.2 shows these modes of communication in graphical form comparing these modes of communication effectiveness of communication versus richness of communication [62]. In figure 2.2, there are two groups, one for modeling and another when we are documenting. Communication modes in modeling includes face to face using white board, face to face conversation, video conversation, phone conversation and e-mail conversation. From figure 2.2, it is clear that face to face communication is most effective and richness than any other communication modes for modeling. When you are documenting then communication modes are video tape, audio tape and paper. From the figure, it is shown that video tape is more effective and richness than any other mode for documenting. The choice of communication modes depends on various condition. As video conferencing may be most effective for communication between distributed team and face to face communication is best when the team are in same office. As we know that requirements are changing in nature in agile methods. Team strategy changes as the requirement changes so the communication mode that well worked well yesterday may not work well tomorrow.

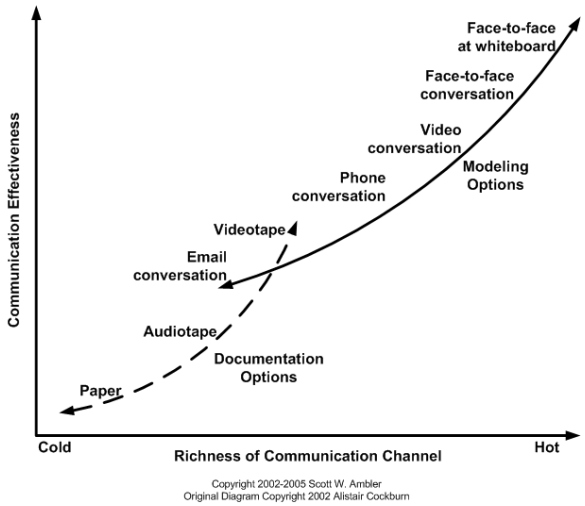

Figure 2.2 Modes of Communication [25]

There are various factors which affect communication between team members. These include physical, temporal and amicability [61]. Physical proximity is the physical difference between development teams. If the teams are collocated or in the same

office then there is richness in communication and if teams are distributed than there will be less richness in communication between teams. Temporal proximity includes time difference between two teams such as they are in different time zone. And last factor is amicability i.e. willingness of one person to communicate with another. The technology used in communication also affects communication. There are various tools and technologies are available in market such as collaborative tools, discussion tools etc. Communication can be in oral form which includes face to face, telephone and video conferencing or in written form which includes emails, letters and reports.

Agile software development lifecycle can be divided into three phases [63]. The primary phase deals with information gathering and understanding the requirement. In mid phase, requirements are analyzed. Requirement may contain ambiguity, so communication at this stage becomes necessary. In end iteration, feedbacks are taken on working software. Researchers has conducted a survey for on the modes of communication that which mode is preferred for communication in different phase [63]. Results show that, in primary level email is a preferred mode of communication between development team and customer. And face to face communication is preferred mode of communication with-in team. Same results are gain in mid level. In end iteration face to face communication is preferred for communication in both with customer and within team. Table 2.2 summarizes these preferred modes of communication.

Table 2.2 modes of communication at different level[63]

Level	**With customer**	**With team**
Primary	Email	Face to face
Mid	Email	Face to face
End	Face to face	Face to face

Face to face communication is preferred in end iteration so that quick decision can be made and to avoid delay in further planning. In agile methods, communication can be active and passive. Active communication is one in which instant reactions of other participants can be observed while in passive communication, instant reaction is not

available. Table 2.3 summarizes the difference between active and passive communication used in agile.

Table 2.3 Active versus Passive Communication Modes

Active Communication mode	Passive Communication Mode
Participant should be present at time of communication	Presence of participant is not required at time of communication
Instant reaction of other team members or customer can be observed	Immediate response is not available
Communication expenses are high	Communication expenses are low
Example:- face to face, video conferencing	Example:- e-mail

In global software development using agile also, communication is one of the main challenge [60]. This communication challenge can be considered as a problem because it can be due to distance in location, temporal distance and different culture[64, 65]. There are 13 issues have been found related to communication in agile and these issues are summarized in table 2.4 [60].

Table 2.4 Issue related to communication

1	Different culture
2	Different language
3	Different working hours
4	Different project background
5	Lack of frequent face to face contact
6	Lack of trust
7	Lack of customer involvement
8	Lack of commitment
9	Miscommunication of requirement

10	Low quality of telecommunication
11	High communication cost
12	Unprepared communication tools
13	**Poor communication infrastructure**

Communication plays an important role in requirement gathering and analysis in agile software development methods. There are some evidence which shows that requirement engineering process in agile methods differs from traditional development methods[66]. It has been identified that most important of the requirement specification approach is communication between developer and customer. Requirements are described as user stories in many agile approaches and each story is composed of three parts [67]. First part of user stories is cards. Requirements are written on cards in brief. And these card also contains history (what work has been done till date), plan (estimation) and goal (to be achieved) of user story. Second part is conversation. These written stories are need to understand by on-going conversation between development teams and customer. And the third part is confirmation, which is the acceptance criteria of story. The structure of the software is communicated through pair programming [59]. Some researchers have studied about communication memory and story card and one of their aim was to study that how requirements are continuous through communication and collaboration [66]. A communication model for requirement engineering was proposed by Abdullah and Honiden [68]. This is mainly based on three main dialogues. First is external stimulus which is used to detect specific requirements from user story card. And two other dialogues were internal stimulus (identifying the goals) and focus. Communication plays vital role in requirement engineering of agile projects. It consists of three activities: gathering, clarifying and evolving [69]. In requirement gathering, teams gather requirements from customer and generate a new story card. In clarifying, team members communicate with each other to clarify the ambiguous requirement and evolving of requirements includes change in existing requirement. Some evidence shows that two types of communication patterns are identified in different phases of requirement engineering process:-one of them is the pattern of communication between story cards, dialogues of team members and memory of previous situations. The other is the pattern of communication between dialogues of team members and

memory of previous situations[69]. Investigation of these communication patterns was carried out which shows that if team members can relate dialogues to story cards and if they can talk about goal related to story card, then this implies that a common notion about the story cards has been set-up among them. If the team members continuously support to each other's dialogues and display evidence of understanding, then this implies that they are aware of others' goals [6].

Face-to-face communication is an essential characteristic of agile processes. However, the desire of organizations to outsource jobs to foreign markets to reduce production costs does not fit well into the scope of agile methods because communication among distributed teams is a challenge in agile. Maintaining good communication can allow distributed teams to keep a single view of the project.

Chapter 3
Problem Statement and Objective

Objective of the thesis is to propose a framework for solution of communication problem in agile. The proposed framework will enable all the module to know its predecessor and successor modules. So whenever there is any change request in any module, all of its predecessor and successor module can be informed. Thus it will provide an effective way of communication among teams.

For above purpose, we have to identify dependency among modules, and make a dependency list of modules. This dependency list will be used to identify other dependent modules. So our objective is to :

"Identify Steps for Making A Dependency List From Given Data Flow Diagram"

It can be achieved into two steps:

1) Identifying steps for converting a DFD into PERT chart

2) Converting PERT Chart into Dependency List

Chapter 4

Work Done

4.1 Steps for DFD to PERT chart

1. Refine DFD up-to level which shows all activities and entities
2. Identify and list down all activities
3. Do numbering of all activities and sub-activities (ex:- 1.a where 1 is activity and 'a' is sub-activity within 1)
4. Arrange in tabular manor.

Table 4.1 Tabular Manner

Name	Activity Number	Input Activity	Output Activity

a. Input activities are those activities whose output is necessary in current activity to perform some action/task. Generally known as predecessor.

b. Output activity column represent successor activity.

5. From above table, build a PERT chart.
6. Those activities, whose output node column is empty, will be organized as end node.
7. Those activities, whose Input node column is empty, will be organized as start node.
8. All end node activities will have a common end point.

4.2 PERT chart to Dependency List

This dependency list for current module shows all the modules dependent on current module (Successor Module) and modules on which current module depends (Predecessor Module). So that if there is any change in current module, all related module can be informed because change may affect both types, input and output .

1. List down all activities in single column

2. A module will be in adjacent list of the current module if

- Module is predecessor of current module
- Module is successor of current module

4.3 Working

For PERT chart to Dependency list, a framework is created. Working of framework is described as:-

1. Initially we have a module number and table developed during previous phase.
2. For each module, framework will ask for number of module in its predecessor list and successor list.
3. Now we will enter module number in predecessor list and successor list for the current module.
4. When the processing for current module is finished, framework will ask same for another module.
5. When all the modules have been inserted successfully, then we can choose another option for change request.
6. After that we have to insert the information being changed and the contact information so that other can be informed about the change.
7. And at the end, a text file is created containing the information. This file can be send using email.

Chapter 5

Results

5.1 Case Study: Order Processing System

5.1.1 DFD to PERT Chart

A. Refine DFD up-to level which shows all activities and entities

For converting into PERT chart we have to follow these steps:-

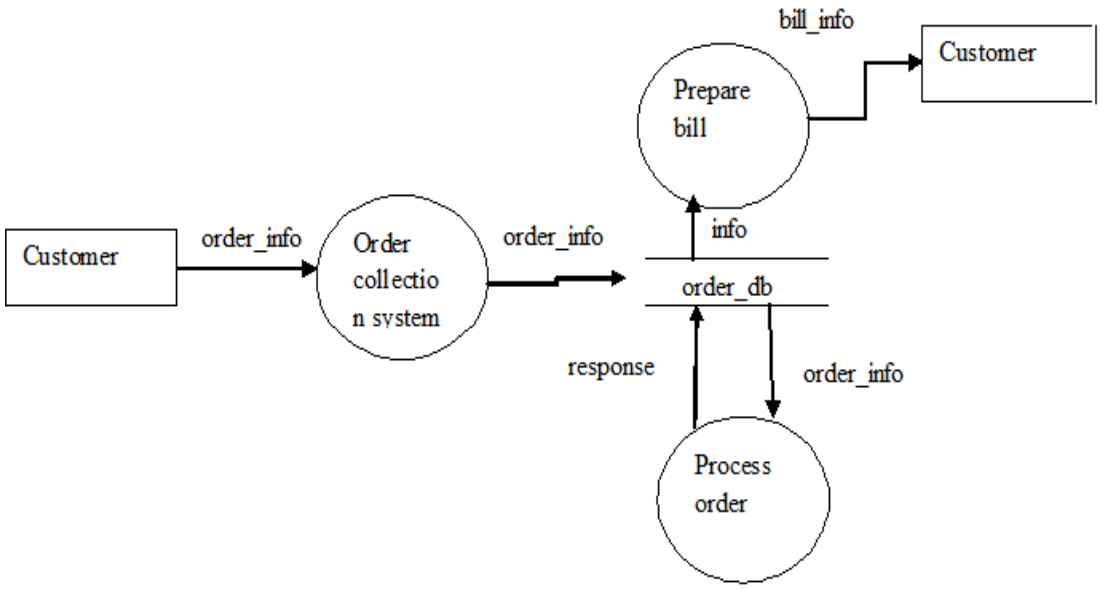

Figure 5.1 : Case Study of Order Processing System

B. Identify all activities and do numbering of all activities

1. Customer gives order:- order_ info.
2. Order collection system collects order.
3. Order information is stored in order database.
4. Process order system will take the order information and give response to order database.
5. Based on information, bill is prepared.
6. Invoice information is given to customer.

C. Arrange in tabular manor

Table 5.1 : Activity Table of Case Study 1

Activity Number	Input Activity	Output Activity
1	0	2
2	1	3
3	2	4
4	3	5
5	4	6
6	5	0

D. From above table, build a PERT chart.

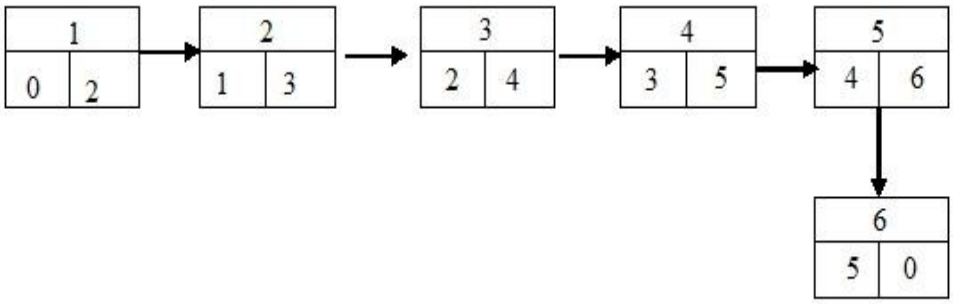

Figure 5.2 : PERT Chart of Case Study 1

5.1.2 PERT Chart to dependency list

For each current module, other modules will be in dependency list if

- Module number is in predecessor part of current module.
- Module number is in successor part of current module.

For performing above work, a framework is developed which will organize the dependency list of each module. Working is shown below in several snapshots.

a) Framework

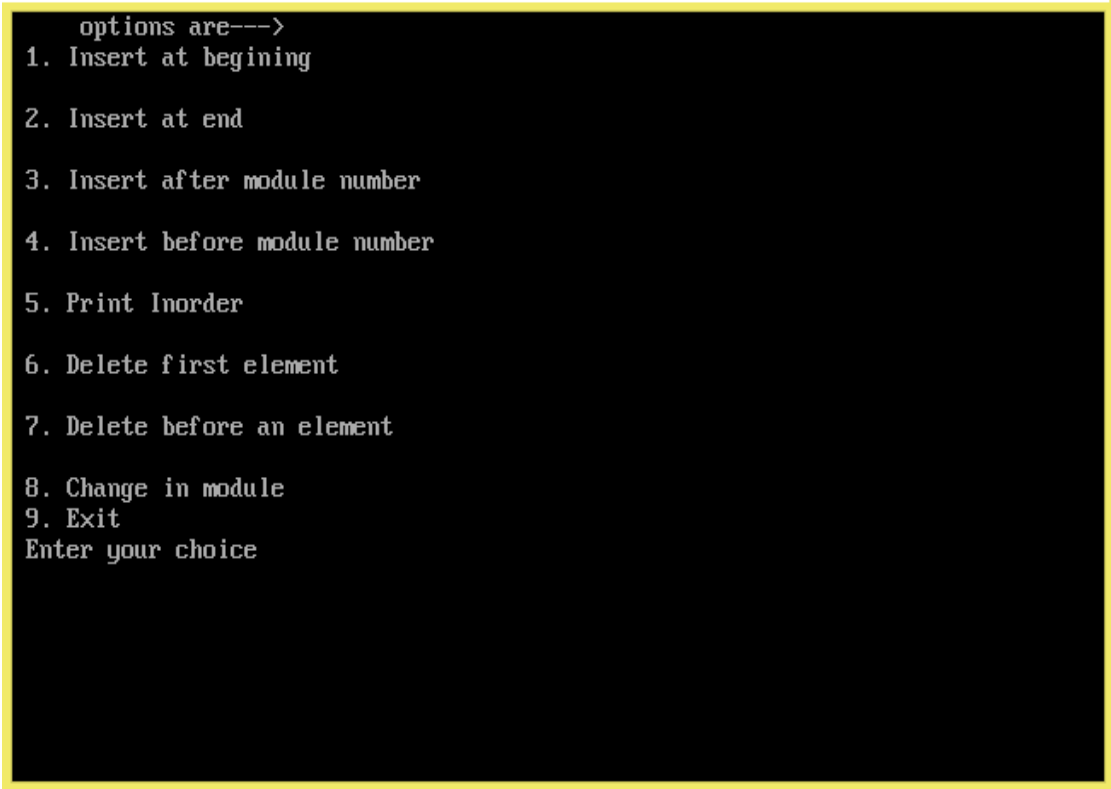

Figure5.3 Framework

b) Enter Information

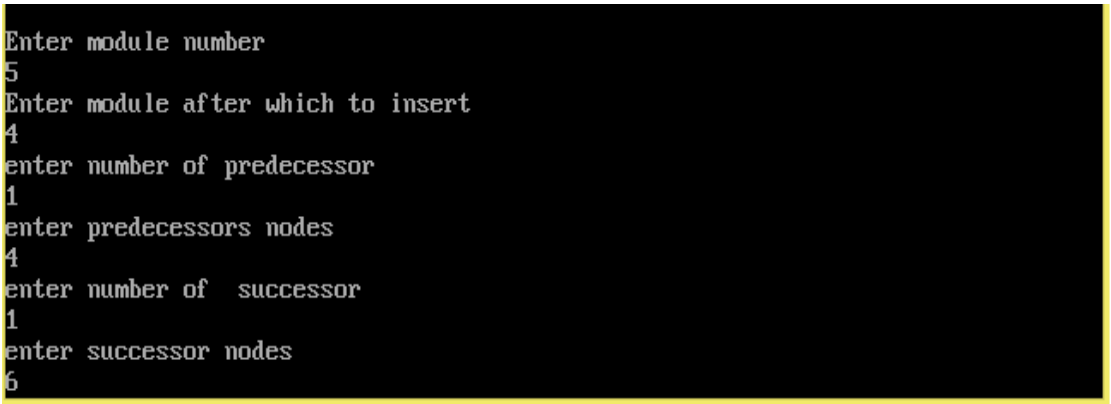

Figure 5.4 Module Info

c) Module information

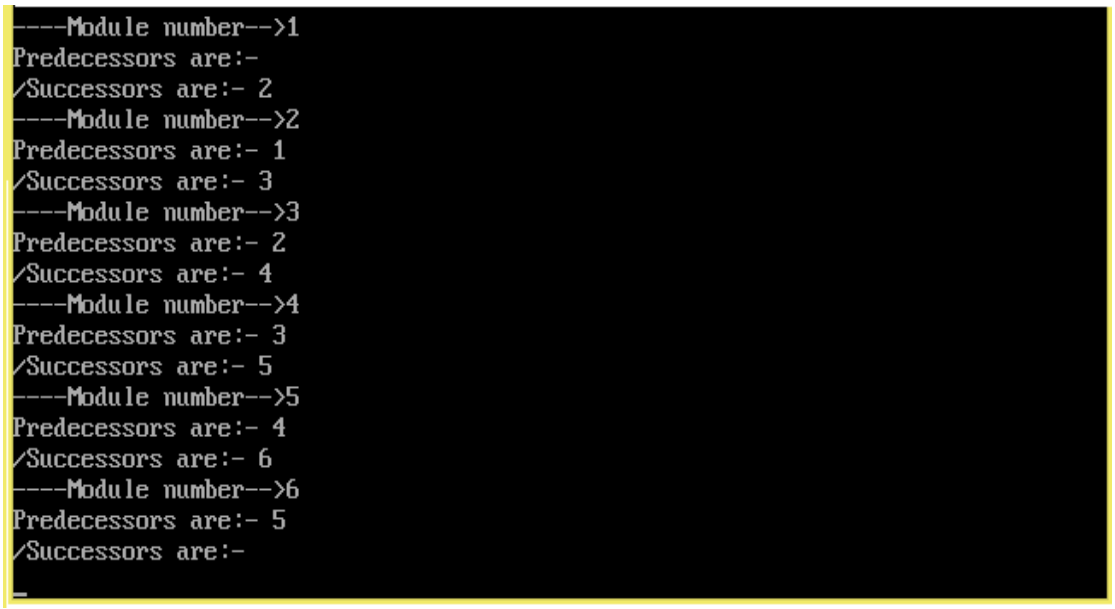

Figure 5.5 Showing informations of related modules

d) Change process

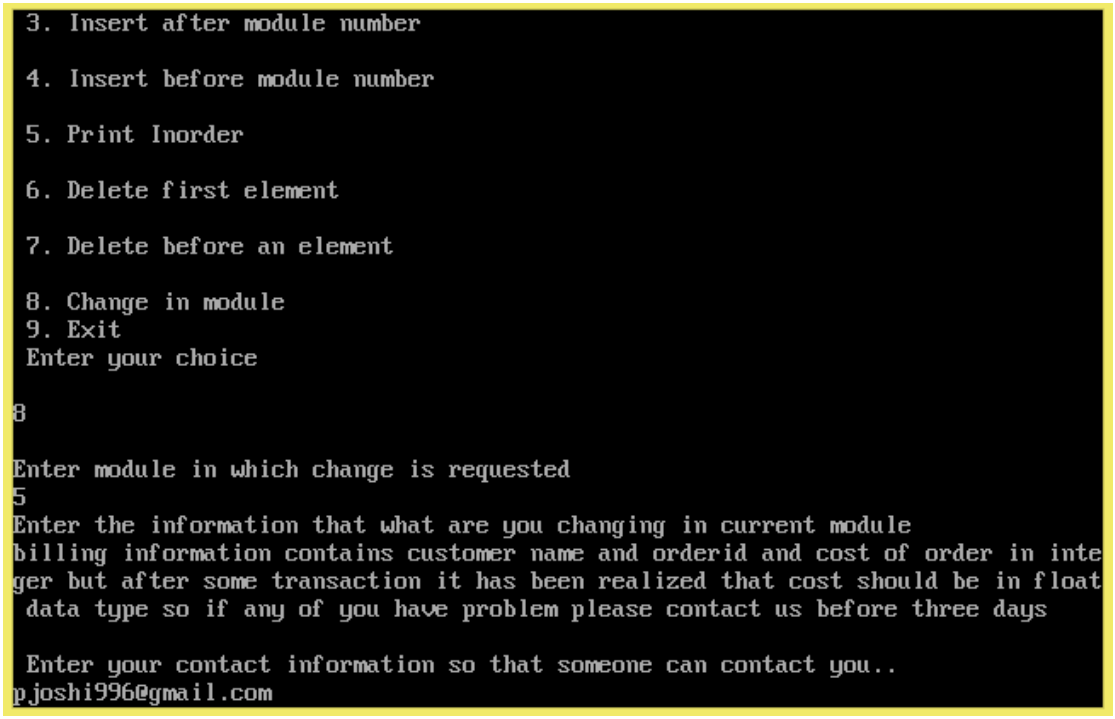

Figure 5.6 Change Requesting

e) Show information module

```
Please inform the team leader who are doing following module

module number:--
 4  6
Message is as follow
billing information contains customer infomation oand orderid and cost in intege
rbut after some transaction it has been realized that cost should be in float da
ta ype so if you have problem please contact us before three days


You can contact on
pjoshi996@ganil.com

File is ready..
 you can send it through e-mail
```

Figure 5.7 Showing related information

From Above Case study

1) Number of modules:- 6
2) Number of change request:- 1
3) Number of module effected:- 2

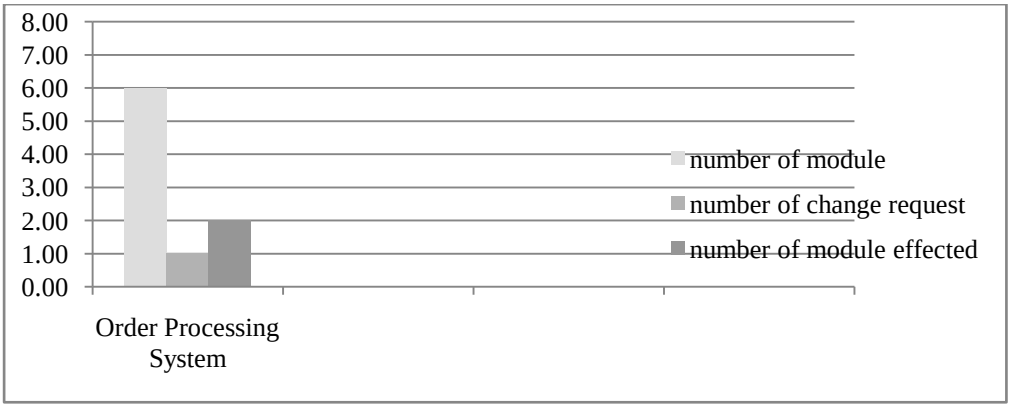

Figure 5.8 : Graphical Representation of Data

5.2 Case Study: Admission Management System

5.2.1 DFD to PERT Chart

For converting into PERT chart we have to follow these steps:-

A. Refine DFD up-to level which shows all activities and entities

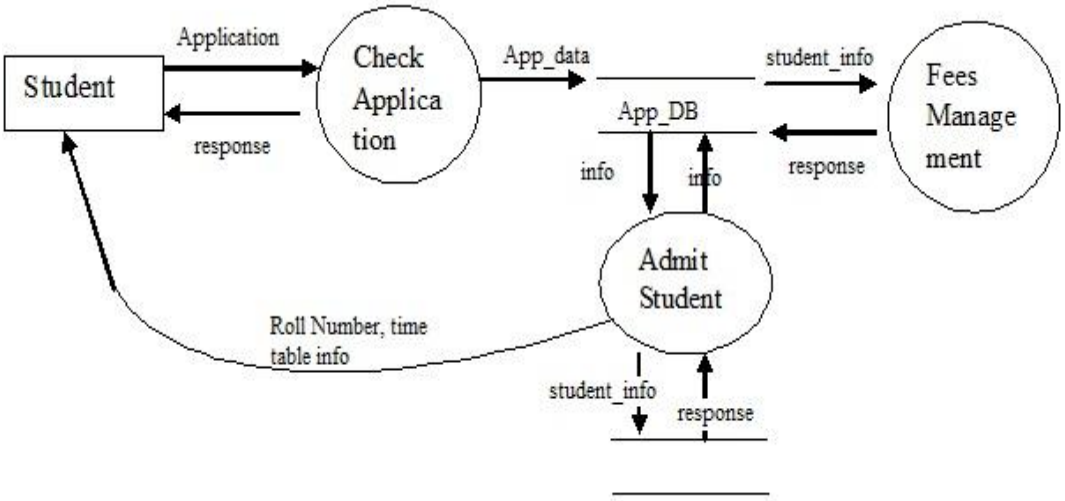

Figure 5.9 Case study of Admission Management System

B. Identify all activities and do numbering of all activities

1. Check application module accepts application and give response to students.
2. Application data is stored in App_DB.
3. Fees management system uses this information and give response that whether the student has submitted fees or not.
4. After payment of fees, admission module will extract the information of students and will enroll those students who have submitted fees.
5. A response is sent to successfully enrolled students.

C. Arrange in tabular manor

Table 5.2 : Activity Table of Case Study 2

Activity Number	Input Activity	Output Activity
1	0	2
2	1	3,4,5
3	2	2

4	2,3	2,5
5	4	0

D. From above table, build a PERT chart.

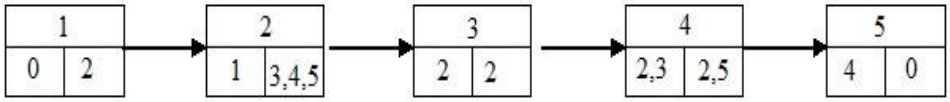

Figure 5.10 PERT Chart of Case Study 2

5.2.2 PERT Chart to dependency list

For each current module, other modules will be in dependency list if

i. Module number is in predecessor part of current module.

ii. Module number is in successor part of current module.

For performing above work, a framework is developed which will organize the dependency list of each module. Working is shown below in several snapshots.

a) Framework

Figure 5.11 Framework

b) Enter Information

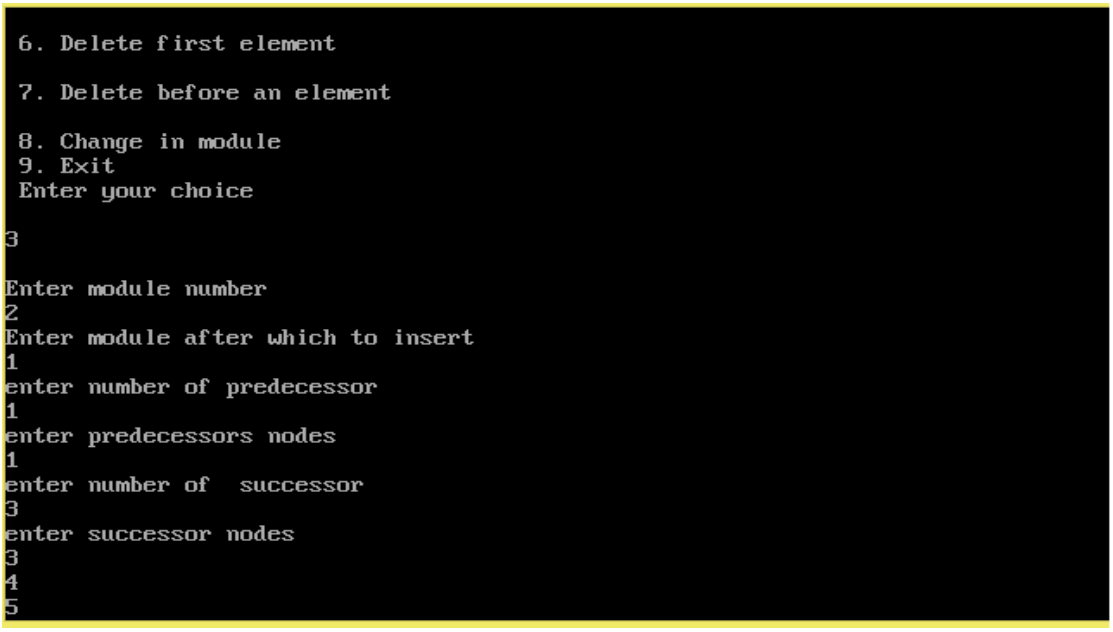

Figure 5.12 Module Info

c) Module information

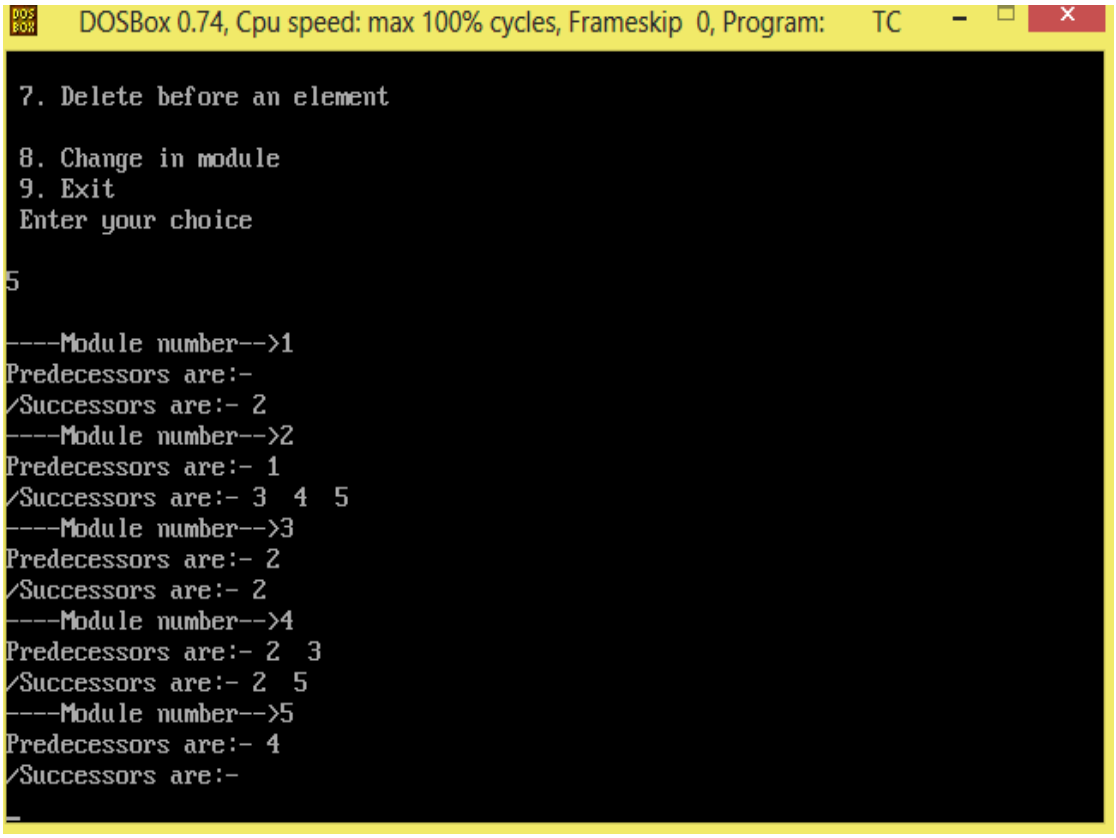

Figure 5.13 Showing informations of related modules

d) Change process

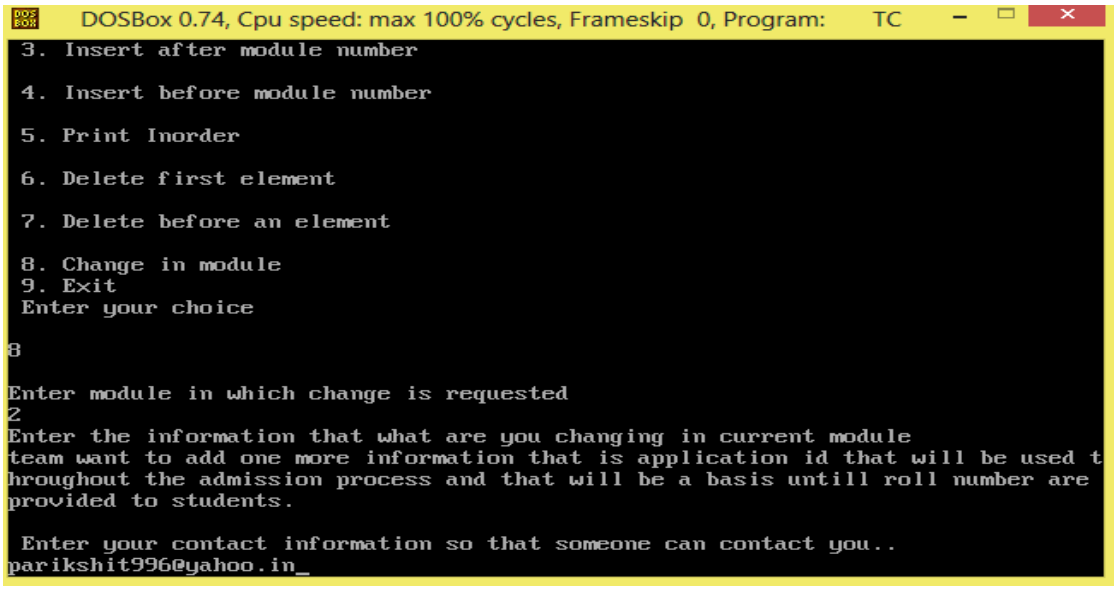

Figure 5.14 Change Requesting

e) Show information module

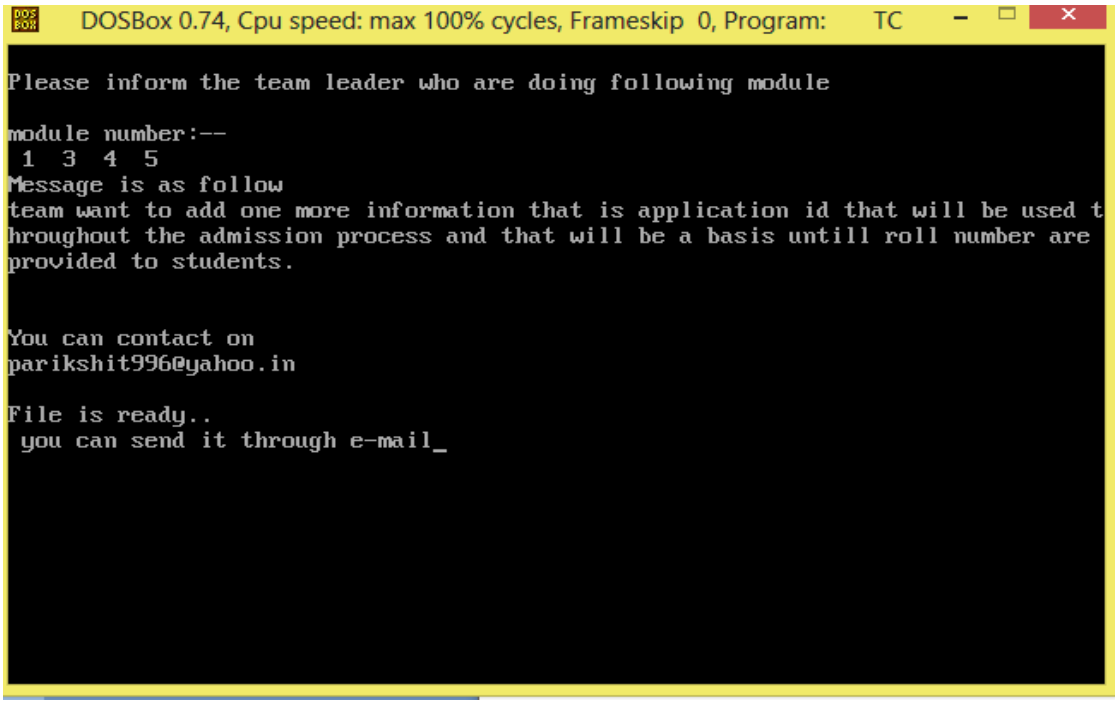

Figure 5.15 Showing related information

From Above Case study

4) Number of modules:- 5
5) Number of change request:- 1
6) Number of module effected:- 4

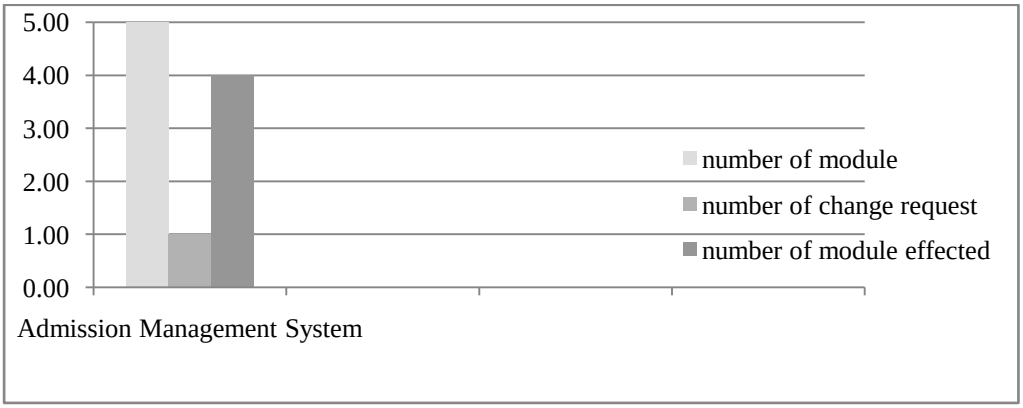

Figure 5.16 : Graphical Representation of Data

Chapter 6

Conclusion and Future Scope

6.1 Conclusion

In today's fast growing business environment, agile software development methods have significant importance for implementing software projects. The field of agile development methods is flooded with innumerous benefits but the organizations using agile methods also face various challenges. Communication is the issue which is faced by many organizations. An effective communication is necessary for developing a successful project using agile models and for increasing satisfaction level of customer. This communication problem consists of several issues in itself such that they may affect organization.

A frame work is used to improve the communication limitation of agile methods. In large organizations, where various distributed teams are working on different module of same project using agile methods, communication is difficult to achieve. This framework tries to solve the communication issue in such a way that if there is any change request than that can be informed to other modules.

6.2 Thesis Contribution

In this thesis a comparison is made between different agile models by which we can chose the model best suited to our requirement and along with a wide knowledge of different agile models is provided. This thesis also contributes in addressing the different issues of communication problem in agile methodology and also proposes a framework which will enable agile teams to communicate with each other to overcome communication issue.

6.3 Future Scope

In future, the limitations of the framework can be improved and some effort can be put to

- To make framework online.
- To add more information about a module such as name of team owner etc.
- To send the generated file over mail using framework.

References

[1]. A. Fruhling and G. Vreede, "Field experiences with extreme programming: Developing an emergency response system," *Journal of Management Information Systems*, Vol. 22, No. 4,2006, pp. 39-68, 2006.

[2]. H. Altarawneh and A. El Shiekh, "A Theoretical Agile Process Framework for Web Applications Development in Small Software Firms," *Sixth International Conference on Software Engineering Research, Management and Applications*, Prague, Czech Republic, 2008.

[3]. P. Joshi, A. Agarwal and S. Goel, "Agile Software Process Model: A Comparative View," *International Conference on Advances in Management and Technology (iCAMT - 2013)*, Proceedings published in *International Journal of Computer Applications (IJCA) (0975 – 8887)*, 2013.

[4]. B. Boehm and R. Turner, "Using risk to balance agile and plan-driven methods," *IEEE Computer*, Vol. 36, No. 6, pp. 57-66, 2003.

[5]. F. P. Brooks ," The mythical man-month Reading," *Addison-Wesley*, 1995.

[6]. C. Juyun, "Issues and challenges of agile software development with Scrum," *Issues in Information Systems*, Vol. IX, No. 2, 2008.

[7]. M. Coram and C. Bohner, "The Impact of Agile Methods on Software Project Management," Proceedings of the 12th *IEEE International Conference and Workshops on the Engineering of Computer-Based Systems* (ECBS'05).

[8]. F. Paetsch , A. Eberlein, and F. Maurer, "Requirements Engineering and Agile Software Development," Proceedings of the 12th *IEEE international Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pp. 308 –313, June 2003.

[9]. K. Beck, "EXtreme Programming Explained," Publisher: *ddisonWesley*, pp. 55-60, 2000.

[10]. Pressman, "Software Engineering, a practitioner's approach," pp.64-69,(2010, 6/e)

[11]. S. Henninger, 1997. "An evolutionary approach to constructing effective software reuse repositories," ACM Transactions on Software Engineering and Methodology, vol. 6, Available at: http://dl.acm.org/citation.cfm?id=248242.

[12]. J. Highsmith, "Agile Software Development Ecosystems," Publisher: *Addison–Wesley*, Boston, MA. ISBN:0-201-76043-6, 2002.

[13]. V. Basili and D. Rombach, "Support for comprehensive reuse," *UMIACS-TR-91-23, CSTR- 2606*, Department of Computer Science, University of Maryland at College Park, 1991.

[14]. K. Beck, "Embrace change with extreme programming," *IEEE Computer* ISBN-13**:** 978-0201616415, Page70–77, 1999.

[15]. A. Cockburn and J. Highsmith, "Agile software development: The business of innovation," *IEEE Computer 34*, 120-122, 2001, DOI=10.1109/2.947100 http://dx.doi.org/10.1109/2.947100.

[16]. J. Highsmith, K. Orr, and A. Cockburn, "Extreme programming," in: *E-Business Application Delivery*, pp. 4–17, 2000.

[17]. M. Poppendieck, "Lean programming," 2001. Available at: http://www.agilealliance.org/articles/articles/LeanProgramming.htm.

[18]. W.W. Royce, "Managing the development of large software systems: Concepts and techniques," in: *Proc. WESCON*, 1970, pp. 1–9, 1970.

[19]. K. Schwaber, "Controlled chaos: living on the edge," Advance development methods Inc., 1996.Available at : http://www.agilealliance.org/articles/articles/ap.pdf.

[20]. The C3 Team, 1998, "Chrysler goes to extremes," in: *Distributed Computing*, pp. 24–28.

[21]. D. Turk, R. France and B. Rumpe, "Limitations of agile software processes," in: Proc. *3rd International Conference on eXtreme Programming and Agile Processes in Software Engineering—XP, 2002*.

[22]Agile manifesto, available at: http://www.drdobbs.com/open-source/the-agile-anifesto/184414755

[23] A. Cockburn, "Agile software development joins the 'would-be' crowd," *Cutter IT Journal*, Vol-15, no-1, 6–12, 2002.

[24] A. Cockburn, "Selecting a project's methodology," *IEEE Software* 17 (4), Volume 17 Issue 4, 64–71, doi 10.1109/52.854070, 2000.

[25] R. Glass, "Agile versus traditional: Make love, not war," *Cutter IT Journal*, Vol. 14, No. 12, Page no 12–18, 2001.

[26] P. Coad, J. deLuca, and E. Lefebvre, "Java Modeling in Color with UML," Publisher : *Prentice Hall*. ISBN-13: 978-0130115102, 1999.

[27] K. Beck, A. Cockburn, R. Jeffries and J. Highsmith, "Agile manifesto," 2001. Available at : http://www.agilemanifesto.org.

[28] S. Ambler, "Examining the Agile Manifesto," *EBG Consulting*, ISBN 978-0985787905, 2012.

[29] A. Cockburn and J. Highsmith, "Agile software development: The people factor," *IEEE Computer* ,ISSN 0018-9162, Page no 131–133, DOI 10.1109/2.963450, 2001.

[30] A. Cockburn and L. Williams, "The costs and benefits of pair programming," in: Proc. *extreme Programming and Flexible Processes in Software Engineering—* XP2000,. ISBN:0-201-71040-4, 2000.

[31] C. L. Mark, "The 12 Agile Principles," available at: http://www.dummies.com/how-to/content/the-12-agile-principles.html

[32] K. Schwaber and M. Beedle's, "Agile Software Development with Scrum," ISBN-13:978-0130676344, pp. 123-125, 2002.

[33] E.J. Chikofsky and J.H. Cross II, " Reverse Engineering and Design Recovery: a Taxonomy," *IEEE Software*, Vol-7, No-1 ,Page no 13-17. DOI=10.1109/52.43044 http://dx.doi.org/10.1109/52.43044, 1990.

[34] I. Sommerville, "Software Engineering," *Addison-Wesley*, ISBN 978-0137035151, Chapter 18, 2004.

[35] J. Highsmith , "Adaptive Leadership: Accelerating Enterprise Agility," Oct. 2000

[36] A. Cockburn, "Agile Model Driven Development," *EBG Consulting*, ISBN 978-0985787905, 2003.

[37] E.. Mnkandla, "A selection framework for Agile Methodologies," 5th International Conference, XP 2004, Garmisch-Partenkirchen, Germany, ISBN 978-3-540-24853-8, June 6-10, 2004.

[38] VersionOne Inc., 2008, "3rd Annual urvey: 2008, The State of Agile Development," Available at: http://www.versionone.com/pdf/3rdAnnualStateOfAgile_Report.pdf.

[39] S. Peter ,"Prioritizing the Product Backlog," 2008. Available at: http://agilesoftwaredevelopment.com/blog/peterstev/prioritizing-product-backlog.

[40] K. Jochen, "Agile Portfolio Management," [E-book] *Microsoft Press*, ISBN 0735625670, 2008.

[41] S.W. Ambler, "Introduction to Agile Modeling," 2002. Available at: www.agilemodeling.com/shared/AMDDIntroduction.pdf .

[42] H. Takeuchia and I. Nonaka, "The New Product Development Game," *Harvard Business Review*, retrieved June 9, 2010.

[43] F. McCarey, "Rascal: Agile Software Reuse Recommender," *Artificial Intelligence Review*, Volume 24 Issue 3-4, 2005.

[44] A.K. Gahalaut, "reverse engineering: an essence for software re-engineering and program analysis," 2010. Available at: http://www.ijest.info/docs/IJEST10-02-06-131.pdf

[45] *M* Poppendiec and T. Poppendieck, "Lean Software Development: An Agile Toolkit", *ISBN: 0321150783,* 2013.

[46] C. W. Krueger, "Software reuse," *ACM Comput. Surv.* 24, 2 (June 1992), 131-183. DOI=10.1145/130844.130856.

[47] M. A. Awed, "A Comparison between Agile and Traditional Software Development Methodologies," *School of Computer Science and software Engineering*, The University of Western Australia, 2008.

[48] B. Boehm, "Get ready for agile methods, with care," IEEE *Computer* , vol.35, no.1, pp.64,69, Jan 2002.

[49] A. Mahanti, "Challenges in Enterprise Adoption of Agile Methods - A Survey," *Journal of Computing and Information Technology*, Vol.14 No.3, 2006.

[50] C. Larman, "Agile and Iterative Development: A Manager's Guide," Boston: *Addison Wesley*, ISBN 978-0131111554, 2004.

[51] S. Sharma, D. Sarkar and D. Gupta, "Agile process and methodology A conceptual study," *International Journal on Computer Science and Engineering (IJCSE)* ISSN : 0975-3397 Vol.4 No. 2, 2012.

[52] G. Keefer, "Extreme Programming Considered Harmful for Reliable Software Development," *AVOCA GmbH* White Paper, 2002.

[53] S. Ambler, "Remaining Agile," *AmbySoft* White Paper, 2005. Available at: http://www.agilemodeling.com/essays/remainingAgile.htm

[54] J. Highsmith, "Objections to Agile Development," *Cutter Consortium*, White Paper, 2004. Available at: http://codebetter.com/davidhayden/2005/08/03/objections-to-agile-development-white-paper-rocks.

[55] S. Abufardeh and k. Magel, "Software Internationalization: Crosscutting Concerns across the Development Lifecycle," *International Conference on New Trends in Information and Service Science*, ISBN 978-0-7695-3687-3,pp. 447-450, 2009.

[56] J. Vecchio, “The Tech Adoption Life Cycle,” *Website*, 2005. available at: http://www.fool.com/research/foolsden000825.htm.

[57] G. Moore, “Crossing the Chasm,” *HarperBusiness*, ISBN *978-0060517120*.

[58] M. Lindvall, “Agile Software Development in Large Organizations,” *IEEE Computer*, 2004, Vol. 37, No. 12, pp. 26–34, 2004.

[59] S. Nerur , R. K. Mahapatra and G. Mangalaraj, “Challenges of Migrating to Agile Methodologies” Magazine *communication of acm*, Volume 48 Issue 5, May 2005 Pages 72-78.

[60] N.K. Kamaruddin, N.H. Arshad, and A. Mohamed, "Chaos issues on communication in Agile Global Software Development," *Business Engineering and Industrial Applications Colloquium (BEIAC)*, 2012 IEEE , vol., no., pp.394,398, 7-8 April 2012.

[61] K. Petri, “Extending Software Project Agility with New Product Development Enterprise Agility,” *Software Process: Improvement and Practice - Special Issue on Industrial Experiences*, Volume 12 Issue 6, pp. 541-543, 2007.

[62] S. Ambler, “Communication on Agile Software Projects,” *Discover to deliver*, EBG Consulting, Inc.,1ST edition, ISBN 978-0985787905.

[63] S. Bhalerao and M. Ingle, "Analyzing the modes of communication in agile practices," *Computer Science and Information Technology (ICCSIT)*, 3rd IEEE International Conference on , vol.3, no., pp.391,395, 9-11 July 2010.

[64] M. Cohn and D. Ford, "Introducing an Agile Process to an Organization", *Proceeding of IEEE computer Society*, pp. 74-78, 2003.

[65] J. Shore and S. Warden, "The Art of Agile Development", *O'Reily Publication*, First Edition. ISBN-13: 978-0596527679, 2007.

[66] L. Cao and R. Bhalerao, “Agile Requirements Engineering Practices: An Empirical Study,” *IEEE Software*, Jan./Feb. 2008, vol. 25, no. 1, 60-67, 2008.

[67] M. Griss, “Systematic Software Reuse: Architecture, Process and Organization are Crucial,” *Software Technology Laboratory, HP Laboratories*, Oct. 1996.

Available at: http://martin.griss.com/pubs/fusion1.htm .

[68]A. Binti and S. Honiden, “Communication patterns of agile requirements engineering,” *In proceedings of the 1st Workshop on Agile Requirements Engineering (AREW '11)*, ACM, New York, NY, USA, Article 1, 4 pages, 2007.

[69] R. Selby, "Empirically based analysis of failures in software systems," *IEEE Transactions on Reliability*, vol. 39, 1990.

List of Publications

1. Joshi, P., Agarwal, A. and Goel, S., 2012. "Agile Software Process Model: A Comparative View", International Conference on Advances in Management and Technology (iCAMT - 2013), Proceedings published in International Journal of Computer Applications® (IJCA) (0975 – 8887).

2. Joshi, P., Agarwal, A. and Goel, S., 2013. "Communication Issues in Agile Methodology: A Survey", Published in International Journals of Latest Research in Science and Technology® (IJLRST) (2278 – 5299).