Chapter 1

INTRODUCTION

This chapter presents an overview of heavyweight and agile methodologies. Different types of heavyweight methods are included like Waterfall model, Iterative Enhancement model, Spiral model, V-shape model, Prototyping model and RAD model. A brief history of agile is included and further the chapter describes key features of agile methodology, agile manifesto and different methodologies of agile. One more methodology Lean software development is also included which discusses Kanban through which JIT is achieved. Scope of the proposed work and organizations of the thesis are presented at the end of the chapter. So, in this chapter main task is to generalize the methods and characteristics of heavyweight and agile methodologies. The development of the concepts, issues and practices behind software development using both techniques have also been done.

1.1 Overview of heavyweight and agile methodologies

Heavyweight methods are also known as traditional or plan-driven software development methods. Heavyweight methodologies are considered to be the traditional way of developing software. These methodologies are based on a sequential series of steps, such as requirements definition, solution building, testing
and deployment. Heavyweight methodology is focused on planning everything from the start of a project. It is characterized as very documentation centric with designated responsibilities for the individual software development disciplines including requirements engineering, architecture design, implementation, and quality assurance. These methodologies are plan driven in which work begins with the elicitation and documentation of a complete set of requirements, followed by architectural and high level design development and inspection. Due to these heavy aspects, this methodology became to be known as heavyweight.

There are several heavyweight methods but only six are included in this thesis. Included methods are Waterfall model, Iterative enhancement model, Spiral model, V-shape model, Prototyping model and RAD model. The most prominent is the Waterfall process.

To fit the new cultures of software development new software development techniques have been introduced. Most software companies nowadays aim to produce valuable software in short time period with minimal costs, and within unstable, changing environments. Agile methodologies are thus introduced to meet the new requirements of the software development companies [1].

Producing high quality software products and meeting the stakeholder’s requirements are major challenges in software engineering. Dynamically changing environments make changes to requirements in the software development process
an inevitable task. Poor requirements and changes to requirements are major elements that cause software project failures [2]. Nowadays the survival of the company is dependent on time-to-market, better customer satisfaction and deliver on time, within costs software. The scientific literature is full of examples in which the success of projects drive the success of companies, or, the other way around, the failure of a project make the company out of business [3]. Worldwide, it is hard to say how many software projects failed or how much money is wasted every year in the failure of software.

Over the past few years software development organizations have learned about the benefits of agile methodologies, such as Scrum, Dynamic system development systems, Feature driven development and Extreme programming. There are several success stories highlighting the benefits of organizations which successfully adopted agile practices. As a result, many organizations are now adopting agile practices for their benefits.

1.2 Different types of heavyweight methods

There are several heavyweight methods present today but this thesis present six of them.
1.2.1 Waterfall Model

In 1960’s “code and fix” was the method employed by software developers. Waterfall model’s first phase tries to capture “What the system will do”, i.e. its system requirements. The second phase determines “How it is designed”. The third phase is of coding. Fourth is testing and final is implementation task, such as training and heavy documentation.

Waterfall is a rigid model where all development activities are planned at the beginning of the project. This model recommends that software should be developed in successive phases. Each phase of the software development process needs to be complete before the staring the next phase. At the end of each phase, an artifact in documented form is produced. At the end of the development cycle, the customer receives the entire product [2]. Since the Waterfall model is the classical model of software engineering. This model is one of the oldest models and is widely used in government projects and in many major companies. As this model emphasizes planning in early stages, it ensures design flaws before they develop. In addition, its intensive document and planning make it work well for projects in which quality control is a major concern [4]. It has five phases (Fig. 1.1).
A requirement definition document is prepared by taking requirements from the customer. The requirements definition document is in a natural language. It serves as a written contract between the customers and the development team. Specifications are derived from the requirements definition document. The specifications are in technical terms which can be understood by the designers. The design phase is next in the Waterfall model, and its inputs are taken from define or analysis phase. The code or design stage is implemented with appropriate programming language to generate source code. The development team tests the source code to validate whether the requirements are satisfied in the software product. Since this process follows a sequential approach, changes made in any
stage affect the other phases of the development process. Despite the problems identified, the model is still widely used in software industry; some researchers are even convinced that it will be around for a much longer period of time. The Waterfall model is connected to high costs and efforts [4]. Hence, Waterfall model has following drawbacks:

- High effort and costs for writing and approving documents for each development phase.
- Extremely hard to respond to changes.
- When iterating a phase the iteration takes considerable effort for rework.
- When the system is put to use the customer discovers problems of early phases very late and system does not reflect current requirements.
- Problems of finished phases are left for later phases to solve.
- Management of a large scope of requirements that have to be baselined to continue with development.
- Big-bang integration and test of the whole system in the end of the project can lead to unexpected quality problems, high costs, and schedule overrun.
- Lack of opportunity for customer to provide feedback on the system.
- The waterfall model increases lead-time due to that large chunk of software artifacts have to be approved at each stage.
Waterfall model can be used when following conditions are satisfied:

- Requirements are fully known and are not going to change anyway.
- This model is used for beginners.
- Developer has lot of time for documentation.

1.2.2 Iterative Enhancement Model

The problems with the Waterfall model created a demand for a new method of developing systems which could provide faster results, require less up-front information, and offer greater flexibility. With iterative development, the project is divided into small parts. This allows the development team to demonstrate results earlier on in the process and obtain valuable feedback from system users [5].

Incremental development is a refinement of the Waterfall development model. The model combines the waterfall life cycle with iterative enhancement (Fig 1.2). Incremental development involves building and validating a subset of the requirements instead of the complete requirements at once [2].
On completing the requirement definition, architectural design and implementation, the program is tested a number of times. There are incremental builds and in each build a subset of product is delivered before the whole project (Fig. 1.2). The Incremental model does not permit developers to implement changes to the requirements. During the first requirement analysis phase, customers and developers specify as many requirements as possible and prepare a SRS document. Developers and customers then prioritize these requirements. Developers implement the specified requirements in one or more iterations. Implementation and testing is based on defined priorities. With iterative development, the project is divided into small parts. This allows the development team to demonstrate results.
earlier and obtain valuable feedback from system users. Iterative model is nothing but a mini Waterfall model.

This model has the same phases as the Waterfall model, but with some restrictions. The phases occur in the same order as in Waterfall model, but these may be conducted in several cycles, with each release providing additional functionality [6].

The aim of the Waterfall model is the delivery of a complete, operational and good quality product. In contrast, Iterative Enhancement model does deliver an operational quality product at each release, but one that satisfies only a subset of the customer’s requirements. With this model, first release is available within few weeks or months, whereas the customer generally waits months or years to receive a product using other models [7]. Iterative Enhancement model has several advantages, disadvantages which are illustrated below. Also when Iterative model can be used is also discussed [8] [9]:

- In Iterative model we can only create a high-level design of the application before we actually begin to build the product and define the design solution for the entire product. Later on we can design and built a skeleton version of that, and then evolve the design based on what has been built.
• In Iterative model we are building and improving the product step by step. Hence we can track the defects at early stages. This avoids the downward flow of the defects.

• In Iterative model we can get the reliable user feedback. When presenting sketches and blueprints of the product to users for their feedback, we are effectively asking them to imagine how the product will work.

• In Iterative model less time is spent on documenting and more time is given for designing.

Iterative model has several disadvantages, which can be illustrated below:

• Each phase of an iteration is rigid with no overlaps.

• Costly system architecture or design issues may arise because not all requirements are gathered up front for the entire lifecycle.

When iterative model can be used is discussed below [8]:

• Requirements of the complete system are clearly defined and understood.

• When the project is big.

• Major requirements must be defined; however, some details can evolve with time.
1.2.3 Spiral Model

Spiral model is one of the popular process model used by industry. It is based on evolutionary approach which couples the iterative nature of prototyping with the controlled and systematic aspects of linear sequential model. It was proposed by Bohm in 1988 and is a popular model used for large size projects. It focuses on minimizing risk through the use of prototype. The model is divided into four quadrants, each with a specific purpose. In the first quadrant, different levels of planning are performed i.e. objectives, alternative means to develop product and constraints imposed on the product are identified. In second quadrant, a thorough risk analysis is done and an appropriate prototype is initiated. In the third quadrant, software development products are sequentially completed i.e. activities like feasibility study, design concepts, specifications, design, coding, testing etc. In fourth quadrant, customer and management evaluate the product and provide permission to continue to the next level of the spiral. The numbers of spirals are not fixed and vary from project to project. It goes until complete system is developed and accepted. As spiral moves to its next level, cost and schedule estimates are adjusted to reflect the present status of development.
Summarizing it can be said that the effective SDLC model namely Spiral model can be used only if the project satisfies the following criteria [11]:

- Spiral model is recommended if the creation of prototype is suitable for the project.
- If the risk evaluation and cost evaluation is important for the project, then spiral model is the best choice for it.
- Spiral model is suitable for medium risk projects and high risk projects.
- Spiral model is also preferred when the project has complex requirement.
- If the user is not sure of the project needs or significant research has to be done on the project, then spiral model is used.
There are some disadvantages of using Spiral model which are illustrated below [12]:

- Can be a costly model to use.
- Risk analysis requires highly specific expertise.
- Project’s success is highly dependent on the risk analysis phase.
- Doesn’t work well for smaller projects.

### 1.2.4 V-Shape Model

This model was developed to relate the analysis and design activities with the testing activities and thus focuses on verification and validation activities of the product. As this model relates the analysis and design phase to the testing phase, testing activities are done in parallel [7].

V-shape model demonstrate the phase along with the type of testing at that stage. Phases in V-shape model are given below:

- **Requirement planning**: It determines the system requirements and how the resources of the organization will be allocated to meet them.
- **High level design**: Defines how the software functions are to implement the design.
- **Coding and implementation**: Transforms the algorithms defined during the detailed design phase into software.
• Unit testing: Checks each coded module for errors.

• Integration and testing: Interconnects the sets of previously unit-tested modules to ensure that the sets behave well as the independently tested modules.

High confidence of customer is required for choosing the V-shaped model approach. Since, no prototypes are produced, there is a very high risk involved in meeting customer expectations [12].

It can be seen that the V-model is an extension of the Waterfall process by mapping verification and validation activities to each sequential development step.
The specification of the requirements is verified through acceptance and system integration testing, and the detailed design and coding activities are verified through unit and component testing (Fig 1.4).

When applied to a project for which it is not well suited, V-shape model has some disadvantages or weaknesses which are illustrated below [13]:

- It does not easily handle concurrent events.
- It does not handle iterations of phases.
- The model is not equipped to handle dynamic changes in requirements throughout the life cycle.
- The requirements are tested too late in the cycle to make changes without affecting the schedule for the project.
- The model does not contain risk analysis activities.

When V-shape model can be used is given below [14]:

- The V-shaped model should be used for small to medium sized projects where requirements are clearly defined and fixed.
- The V-shaped model should be chosen when ample technical resources are available with needed technical expertise.
1.2.5 Prototyping Model

A prototype is an early sample or model built to test a concept or process or to act as a thing to be replicated or learned from. A prototype is designed to test and trial a new design to enhance precision by system analysts and users. Prototyping serves to provide specifications for a real, working system rather than a theoretical one [15]. In general a prototype acts as a sample to test the process. From this sample we learn and try to build a better final product. The prototype may or may not be completely different from the final system we are trying to develop. This type of System Development Method (SDM) is employed when it is very difficult to obtain exact requirements from the customer. While making the model, user keeps giving feedbacks from time to time and based on it, a prototype is made. Completely built sample model is shown to user and based on her/his feedback; the SRS (System Requirements Specifications) document is prepared (Fig. 1.5). After completion of this, a more accurate SRS is prepared, and now development work can start by using any of the existing software development models.

Fig 1.5 Prototyping model
When prototype is shown to the user, he/she gets a proper clarity of the functionality of the software and customer can suggest changes and modifications, if any. This approach for developing the software is used for non-IT-literate people. They usually are not good at specifying their requirements. Also when client is not confident about the developer's capabilities, he/she asks for a small prototype to be built. Based on this model, he/she judges capabilities of developer. Prototype making reduces risk of failure, as potential risks can be identified in early stages. Iteration between development team and client provides a very good environment for development.

Disadvantages [16] of prototyping model are:

- Prototyping is usually done at the cost of the developer. So it should be done using minimal resources. It can be done using Rapid Application Development (RAD) tools. Sometimes the start-up cost of building the development team, focused on making prototype, is high.
- Once we get proper requirements from client after showing prototype model, it may be of no use. That is why; sometimes we refer to the prototype as "Throw-away" prototype.
- It is a slow process.
- Too much involvement of client is not always preferred by the developer.
- Too many changes can disturb the rhythm of the development team.
1.2.6 Rapid Application Model

Rapid Application Development model is an incremental software development process model that emphasizes short development cycles. The Rapid Application Development model was proposed by IBM in 1980’s and later on was introduced to software community by James Martin through his book “Rapid Application Development”. The important feature of RAD model is increased involvement of the user/customer at all the stages of the lifecycle through the use of GUI tools. The RAD model is high speed adaptation of the linear sequential model in which rapid development is achieved by using component-based construction. The process starts with the building a rapid prototype and is delivered to the customer for the use and her/his feedback. Once the user or customer validates the rapid prototype after using it, requirement specification document is delivered and design is done to give final shape to the product. After the product is installed, maintenance of the product is continued by refining the requirement, specification, design or coding phase.

RAD is characterized by the quick turnaround time from requirements definition to completed system. It follows a sequence of evolutionary system integrations or prototypes that are reviewed with the customer, discovering requirements along the way the development of each integrated delivery is restricted to a well-defined period of time, usually about 60 days, called a time-
box. Factors that allow a system to be created in the 60 days of the timebox, without sacrificing quality, include the use of high-powered development tools, a high reuse factor, and knowledgeable and dedicated resources [17].

The critical end-user roles shift work from programming and testing to planning and design. More work is created for users at the front of the life cycle, but they are rewarded with the system that is built more quickly.

There are four phases in RAD model and these are:

1. **Requirement planning phase**
   
   Requirements are captured using any group elicitation technique.

2. **User description**
   
   Joint teams of developers and users are constituted to prepare, understand and review the requirements. The team may use automated tools to capture information from other users.
3. Construction phase

This phase combines the detailed design, coding and testing phase of Waterfall model. Here, we release the product to customer; it is expected to use code generators, screen generators and other types of productivity tools.

4. Cut over phase

This phase incorporates acceptance testing by the users, installation of the system, screen generators and other types of productivity tools.

In general it can be said that RAD model is flexible and adaptable to changes and it increases reusability of components. It can handle large projects. Quick initial reviews are present in this model. RAD realizes an overall reduction in project risk and it incorporates short development cycles; as a result development time reduces. Customer feedback is constantly involved in this model.

Disadvantages of RAD model [9] [10] can be discussed in the following ways:

- RAD is not appropriate when technical risks are high.
- Can’t be used for small projects.
- Depends on strong team and individual performances for identifying business requirements.
- Requires more resources and money to implement RAD.
- All applications are not compatible with RAD.
• Only system that can be modularized can be built using RAD.
• Requires highly skilled developers and designers.
• High dependency on modeling skills.
• Need both customer and developer commitments to complete a project.
• Inapplicable to cheaper projects as cost of modeling and automated code generation is very high.

RAD model can be used in the following two conditions [18] [19] [20]:

• RAD should be used when there is a need to create a system that can be modularized in 2-3 months of time.
• It should be used if there’s high availability of designers for modeling and the budget is high enough to afford their cost along with the cost of automated code generating tools.

RAD SDLC model should be chosen only if resources with high business knowledge are available and there is a need to produce the system in a short span of time (2-3 months) [18].

1.3 History of agile

Developing software is a difficult and extremely labor-intensive activity. As with many labor-intensive activities, developing software is error prone. Every year there are more software-based devices controlling functions that are critical to
human survival. The chances of disasters and failures of these software-based devices have greatly increased [21]. Over the past decades, several of these failures resulted in either loss of lives or property. Many of these development failures are attributed to software requirements engineering issues. Worldwide, it is hard to say how many software projects failed or how much money is wasted. If failure is defined as the total abandonment of a project before or shortly after it is delivered, and if one accepts a conservative failure rate of five percent, then billions of dollars are wasted each year on bad software. Some of the software failures have led to significant loss of properties and lives [2].

Incremental software development methods have been traced back to 1957. In 1974, a paper by E. A. Edmonds introduced an adaptive software development process. Concurrently and independently the same methods were developed and deployed by the New York Telephone Company's Systems Development Center under the direction of Dan Gielan. In the early 1970s, Tom Gilb started publishing the concepts of Evolutionary Project Management (EVO), which has evolved into Competitive Engineering. During the mid to late 1970s Gielan lectured extensively throughout the U.S. on this methodology, its practices, and its benefits. Lightweight agile software development methods evolved in the mid-1990s as a reaction against the heavyweight Waterfall-oriented methods, which were characterized by their critics as being heavily regulated, regimented, micromanaged and overly incremental approaches to development. Lightweight agile methods contend that
they are a return to development practices that were present early in the history of software development. Early implementations of agile methods include Rational Unified Process (1994), Scrum (1995), Extreme Programming (1996), Adaptive Software Development, Feature Driven Development (1997), and Dynamic Systems Development Method (1995). These are now collectively referred to as agile methodologies, after the Agile Manifesto was published in 2001 [22].

In February 2001, a group of people kept a meeting in Utah, in order to find an alternative method to the existing heavy software development methodologies and the result was agile methodology. Agile methodologies use minimum documentation, just sufficient for the project to run under good conditions. Agile methodologies and practices are based on iterative enhancements.

1.4 Agile definition and features

In today’s world software is evolving at a very high speed. The complexity of software has increased to a high level but time limit and budget are decreasing. The customer also demands maintenance and quality of software. Requirements cannot be given at the starting and this is the main drawback of heavyweight methods. In heavyweight methods, changing and incomplete requirements creates difficulty for developers. To remove all these agile is the recent methodology.
Earlier to this technology heavyweight methodologies are there which defines a lot of methodologies like Waterfall model, Prototyping model, V- shape model, Spiral model, Iterative enhancement model etc. This research is on agile methodology which itself includes a lot of methodologies. While the heavyweight Waterfall methodologies rely on a large number of documents. Heavyweight methodologies require defining and documenting a stable set of requirements at the beginning of a project. Lightness in case of agile signifies the ability to adapt to the changes, no sequential series of steps are followed and no documentation is required. Agile teams decrease costs as well. They do this partly by technical excellence; the best agile projects generate only a few bugs per month. They also eliminate waste by cancelling bad projects early and replacing costly development practices with simpler ones.

So, agile methodology can be defined as an approach to project management, typically used in software development. It helps teams respond to the unpredictability of building software through incremental, iterative work cadences, known as sprints [23].

This is a value-based software engineering practice and in this type of software engineering the focus is not only on the customer, but concerns all stakeholders involved in the creation of software. “Agile” means nimble or quick moving and “agility” means the ability to think and draw conclusions quickly. In
terms of software development agility is dynamic, content specific, growth oriented and able to adopt the changes quickly and easily. According to Erickson et al. we define agility as: Agility means to strip away as much of the heaviness, commonly associated with the traditional software development methodologies, as possible to promote quick response to changing environments, changes in user requirements, accelerated project deadlines and the like [24].

In agile, software development does not follow a defined process, but uses very short iterations of (2-4 weeks) which focus on producing working software. Agile also allows requirements to emerge throughout the development process. Agile methodology allows changes to requirements even late into the project with minimal impact on software functionality and quality of the delivered product. It has the properties like iterative development having short iterations, working versions at conclusion of each iteration, fully integrated and tested methodology, adaptable i.e. can evolve with each iteration and people-centric i.e. developers & management are equal.

So, key features [25] of agile methodology are as follows:

- **Adaption**

  The teams of developers are self-organized based on the daily meetings. Developers and customers self-organize at the end of every increment to guide the project and create the greatest value.
• **Emergence**

The architecture, team structure, and requirements emerge during the course of the project rather than being determined at its outset. The team preliminary and sketchy vision of requirements and architecture guides the team.

• **Modularity**

Modules are designed to work for a specific task. Agile methods break the whole task into small parts. Every part is solved individually with a very little planning. These tasks do not directly involve future planning. This means the whole concentration is on present module and its requirement at that time. The future scope of that module and what other modules will contain is not of concern at this stage. They are just containing one part and others are included at latter stages.

• **Incremental development**

The life cycle of project is divided into small iterations. Each iteration last from two to four weeks. Each iteration undergoes the whole process of software development life cycle. This includes requirement analysis, planning, design, implementation, unit testing and acceptance testing. After acceptance testing of software one iteration is over. Once it is released it is verified by the stakeholders and corrections are made. In
incremental development, there is small software released with rapid cycles. So the whole application is built in small steps.

- **Team composition**

  Team consists of experienced developers. It is because there is no such documentation for the project. Everything depends on how the stakeholder gives the definition of a particular module. The developer should understand the viewpoint of the stakeholder. This understanding requires a lot of experience, so almost negligible place for fresh candidates.

- **Meetings**

  Meetings provide an internal status of the project.

- **People oriented**

  In agile methodology, instead of the processes and technology, people are favored. Whatever method of agile is followed, each agile team consists of one customer representative. Customer representative is the one who is involved as an end user of the project. This person is appointed by the stakeholders (customer representative who will give money). He acts on their behalf and can make any commitment required to carry on with the development of the project. He is also responsible for answering for questions raised by the developer during any mid iteration i.e. during development. At the end of the iteration this customer representative together with the stakeholders, review the progress of the project and re-
evaluate the priorities of the modules. It is done in order to satisfy the company’s current needs and goals.

- **Contrast with cowboy coding**

  Cowboy coding is the absence of a defined method. Team members do whatever they feel is right. Agile development’s frequent re-evaluation of plans, emphasis on face-to-face communication, and relatively sparse use of documents sometimes causes people to confuse it with cowboy coding. Agile teams, however, do follow defined (and often very disciplined and rigorous) processes. As with all methodologies, the skill and experience of the users define the degree of success and/or abuse of such activity. The more rigid controls systematically embedded within a process offer stronger levels of accountability of the users. The degradation of well-intended procedures can lead to activities often categorized as cowboy coding [26].

- **Cooperative**

  In agile, the customers and the developers work together with close communication. Most agile implementation used to have a daily meeting among the team members. Together with the developers, the customer representative or one of the stakeholders also attends the meeting. This meeting helps everyone to know the actual progress of the project. In this session each team that is handling individual modules is required to report what they have done last day and what they intend to do today. Also they
discuss the difficulty they are facing. This face to face communication helps them to get the solution to their problems.

- **Collaborative and communicating working style**

  When the whole of the team working on a single project is in same location, instead of written documents, agile method to a great extend emphasize on face to face communication between the team members. So, the cost of moving the information between the people is reduced. This communication is facilitated through a single open office of agile team known as bullpen. The team size is typically small, ranging from 5 to 9 people per team. This simplifies the communication and the cooperation among the team members can be increased. It is not necessary that one team work on one module, multiple teams can work on one module. So, more development efforts can be there if they are needed. A prioritization across teams is done in order to achieve things in right manner and at right time. When team works at different locations, then contact is maintained through video conferencing, e-mails etc.

- **Lightness**

  Lightness signifies the ability to adapt to the changes. The numbers of processes that are followed during the development of project are very less. Also, the documentation part of the project takes a back seat. So, it is
not necessary to provide a heavy documentation to the project at the beginning of the development itself. The documentation regarding the project is made available to the developers as and when required. Moreover the documentation is usually replaced by the conversation between the people. The product that is released after iteration contains many bugs but after many iterations the product has full functionalities.

1.5 Agile manifesto

Agile manifesto is collection of values and principles, which are found in most agile methods. First agile alliance was formed when seventeen representatives of different agile methods met to discuss alternative to rigorous, documentation driven software development system.

Agile alliance formulated their ideas into values and further to twelve principles [27] that support values:

1. Highest priority is to satisfy the customer through early and continuous delivery of valuable software.

2. Welcome changing requirements, even late in development. Agile processes harness change for the customer’s competitive advantage.

3. Deliver working software frequently from a couple of weeks to a couple of months, with a preference to 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 to conveying information to and fro 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 constant pace indefinitely.

9. Continuous attention to technical excellence and good design enhances agility.

10. Simplicity which is 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.

Values of Agile manifesto [27] are as follows:

1. Individuals and interactions over processes and tools.

2. Working software over comprehensive documentation.
3. Customer collaboration over contract negotiation.
4. Responding to change over following a plan.

1.6 Agile methodologies

There are several agile methodologies but four of them are explained in this thesis.

1.6.1 Scrum

Scrum methodology was initiated by Ken Swaber in 1995. It was practiced before the announcement of agile manifesto. Later, it was included into agile methodology, since it has the same underlying concepts and rules of agile development. Scrum has been used with the objective of simplifying project control through simple processes, easy to update documentation and higher team iteration over exhaustive documentation [28].

Scrum is huddled mass of players engaged with each other to get a job done. In software development, the job is put out a release. Scrum for software development came out of rapid prototyping community because prototype makers wanted a methodology that would support an environment in which the requirements were not only incomplete at the start, but also could change rapidly during development. Scrum introduces the ideas of flexibility, adaptability and productivity. Scrum concentrates on how the team members should function
together in order to produce the system flexibility in constantly changing environment. The main idea of Scrum is that system development involves several environment and technical values (e.g. requirements, time frame, resources and technology) that are likely to change during the process. This made the development process unpredictable and complex, requiring flexibility of the system development process to be able to respond to the changes [29].

Scrum is an iterative and incremental agile software development framework for managing software projects and product or application development. Its focus is on "a flexible, holistic product development strategy where a development team works as a unit to reach a common goal" as opposed to a "traditional, sequential approach" [30].

The development teams frequently iterate new increments of functionality. Stakeholders/product owners prioritize lists of required systems functionality, cost, timetables, and quality based on emerging business conditions. After the completed iteration, users and development teams collaborate on what to develop next, based on what was just developed and the new business needs. Ken Schwaber, Mike Beedle, Jeff Sutherland and others have contributed significantly to the evolution of Scrum [2]. Over the last ten years in particular, Scrum has gained increasing popularity in the software community due to its simplicity, proven productivity, and ability to act as a wrapper for various engineering practices promoted by other agile
methodologies. Scrum for software development came out of the rapid prototyping community because prototype makers wanted a methodology that would support an environment in which the requirements were not only incomplete at the start, but also could change rapidly during development. It’s most common usage is for managing software development and/or maintenance projects.

1.6.1.1 Terminologies in agile Scrum development

Various terminologies are used in Scrum development. These are [31][2]:

*Product backlog*

An evolving prioritized queue of business and technical functionality that needs to be developed into a system.

*Product owner*

An important stakeholder-demanding role may represent a larger group of products. Ensure business relevance and manages the product content.

*Scrum master*

The person responsible for the Scrum process, make sure that the Scrum process is used correctly and maximizing its benefits. The Scrum master acts as a coach.

*Scrum team*
It is a small cross-functional group of self-organized developers responsible for actual analysis, design, implementation, and testing of the software product. Scrum team is usually not more than ten people.

Test execution

Lists the test planned, actual implemented, failed and passed, and yet to be implemented.

Sprint

Sprints are interactive cycles where the functionality is developed or enhanced to produce new increments. Each Sprint includes the traditional phases of system development. One Sprint is planned to last from one week to one month. There may be 3 to 8 Sprints in one system which is ready for distribution. Also there can be more than one team building the increment. It is an iteration of work during which increments of product functionality are implemented. It is normally two to four weeks cycle.

Sprint backlog

It contains features that the team would implement in the current sprint. These features are from the prioritized list of requirements in the product backlog. The sprint backlog holds tasks the scrum team is currently working on.

Sprint review

Four hours limit period, discussing the product increment, reviewing the work that was completed and not completed.
Sprint planning meeting

It occurs on the first day of the sprint; the sprint backlog content is established.

Daily scrum meeting

Before each sprint, the team plans a sprint. They also decide and reprioritize goals for the next sprint; they select and move features from the product backlog to the sprint backlog.

Sprint burn-down chart

It depicts the total task hours remaining per day. It is easy to track the status of the project progress from the burn-down chart.

Shippable product increment

At end of sprint, work should be potentially shippable, ready in hand to customer, put on store or show to stakeholder. Sprint ends with review and retrospective.

1.6.1.2 Scrum development process

Roles in Scrum are defined as pigs and chickens. Pigs are working or core members. Chickens are ancillary or non-working members. Scrum requires regular coordination among these members. Pigs are Scrum master, Product owner and team. Chickens are Managers and Stakeholders. Scrum methodology includes both managerial and development processes [32].
There are three different phases in Scrum, which are Pre-game phase, Mid game phase and Post game phase [34]. These are described below:

### 1.6.1.2.1 Pre-game phase

The pre-game phase includes two sub-phases: Planning and Architecture/High level design.

**Planning**

Planning phase includes the definition of the system being developed. A product backlog is created containing all the requirements that are currently known. The requirements can originate from the customer, sales and marketing division, customer support or software developers. The requirements are prioritized and
effort needed for their implementation is estimated. The product backlog list is constantly updated with new and more detailed items, as well as with more accurate estimations and priority orders.

Architectural/High level design

In the architecture phase, the high level design of the system including the architecture is planned based on the current items in the product backlog. In case of enhancement to existing system, the changes needed for implementing the backlog items are identified along with the problem they may cause. A design meeting is held to go over the proposals for the implementations and decisions are made on the basis of this review.

1.6.1.2.2 Mid-game phase

The development phase also called Mid-game phase is the agile part of Scrum approach. This phase is treated as “black box” where unpredictable is expected. The different environment and technical variables (such as time frame, quality, requirements, resources, implementation technologies and tools and even development methods) identified in Scrum, which may change during the process are observed and controlled through various Scrum practices during the Sprints, of development phase. In the development phase, the system is developed in sprints.
1.6.1.2.3 Post-game phase

The post-game phase contains the closure of the release. This phase is entered when an agreement has been made that the environmental variables such as the requirements are completed. In this case, no more items and issues can be found nor can any new ones be invented. The system is now ready for release and preparation for this is done during post-game phase, including the tasks such as integration, system testing and documentation.

1.6.1.3 Scrum tools and documentation

Scrum does not focus on creation of excessive documentation. There are various tools available for tracking scrum projects like Microsoft Project, Microsoft Visual Studio, IBM rational have add-ins. One of the simplest among them is an Excel based worksheet which is part of Microsoft Scrum Kit. Microsoft Scrum Kit includes Excel templates for product backlog, sprint backlog, burn down tracking and various charts, which give visual representation of project status for consumption of management. One excel document per sprint is prepared [32].

1.6.2 Extreme Programming (XP)

Extreme Programming (XP), originally described by Kent Beck, has emerged as one of the popular and controversial agile methods. Extreme Programming has evolved from the problems caused by the development cycles of
heavyweight development models. It first started as an opportunity to get job done with practices that had been found effective in software development processes. After a number of successful trials, XP methodology was developed on the key principles and practices used. The term ‘extreme’ comes from taking these commonsense principles and practices to extreme levels.

According to Ron Jeffries Extreme Programming is a discipline of software development based on values of simplicity, communication, feedback, and courage. It works by bringing the whole team together in the presence of simple practices, with enough feedback to enable the team to see where they are and to tune the practices to their unique situation [35] [36] [37].

Extreme Programming is a lightweight method designed for small-to-medium sized team developing software with rapidly changing requirements. It works by bringing together the whole team in the presence of twelve simple practices, which are Planning Game, Short Releases/Frequent small releases, Metaphor, Simple design, Testing first, Refactoring, Pair programming, Collective ownership, Continuous Integration, Coding Standards, On-site Customer, and 40-hour week [38].

XP teams perform nearly every software development activity simultaneously i.e. analysis, design, coding, testing, and even deployment occur with rapid frequency. XP does it by working in iterations week-long increments of work.
Every week, the team does a bit of release planning, a bit of design, a bit of coding, a bit of testing, and so forth. They work on stories, very small features, or parts of features that have customer value. Every week, the team commits to delivering four to ten stories. Throughout the week, they work on all phases of development for each story. At the end of the week, they deploy their software for internal review [39]. XP is a disciplined approach to delivering high-quality software quickly and continuously. XP team members spend few minutes on programming, few minutes on project management, few minutes on design, few minutes on feedback, and few minutes on team building many times each day [40]. It promotes high customer involvement, rapid feedback loops, continuous testing, continuous planning, and close teamwork to deliver working software at very frequent intervals, typically every 1-3 weeks. It consists of small teams usually of size 20. Also it requires onsite customer presence. So, XP recipe is based on four simple values – simplicity, communication, feedback, and courage and twelve supporting practices [36], which are:

- Planning Game
- Small Releases
- Customer Acceptance Tests
- Simple Design
- Pair Programming
- Test-Driven Development
• Refactoring
• Continuous Integration
• Collective Code Ownership
• Coding Standards
• Metaphor
• Sustainable Pace

1.6.2.1 XP principles

A summary of XP principles [41] [42] [43] is shown below:

• **Refactoring**

  Restructure system without changing its behavior to remove duplication, improve communication, and add flexibility and simplicity.

• **Pair programming**

  All code for a production release is created by two people working together at a single computer. XP proposes that two coders working together will satisfy user stories at the same rate as two coders working alone, but with much higher quality. So, two programmers on one machine work hence four eyes are better than two.

  For pair programming the two useful questions that arise are

  1. Isn’t it wasteful to have two people do the work of one?
  2. Isn’t it a risk to reduce the amount of documentation?
Clarifications for the first problem are- In pair programming, two people aren’t really doing the work of one. In pair programming, one person is programming and the other is thinking ahead, anticipating problems, and strategizing.

Clarifications for second problem are- In order to reduce documentation, we have to replace it with some other form of communication. That’s what XP does. Increasing the amount of written communication also increases our risk. What if that information goes out of date. Favoring written communication may decrease our agility, but favoring spoken communication may require more work to disseminate information to the people who need it.

- **Collective ownership**
  
  Anyone can change code anywhere in the system at any time.

- **Continuous integration**
  
  Integrate and build the system many times a day, every time a talk is completed. Development teams must integrate changes into the development baseline at least once a day. This concept is also called continuous integration.

- **40-hour week**
  
  Never work overtime i.e. second week in a row.
• **On-site customer**

  Real, live user on the team, available full-time to answer questions.

• **Coding standards**

  All code written accordance with rules emphasizing communication through the code.

• **Project velocity**

  Velocity is a measure of how much work is getting done on the project. This important metric drives release planning and schedule updates.

• **User story**

  A user story describes problems to be solved by the system being built. These stories must be written by the user and should be about three sentences long. User stories do not describe a solution, but these use technical language. A sample user story might go like this: Search for customers. The user tells the application to search for customers. The application asks the user to specify which customers. After the user specifies the search criteria, the application returns a list of customers meeting those criteria. Because user stories are short and somewhat vague, XP will only work if the customer representative is on hand to review and approve user story implementations.
1.6.2.2 Life cycle of XP

Extreme Programming consists of five phases. Extreme Programming phases are Exploration phase, Planning phase, Iteration phase, Productionizing phase and Maintenance phase (Fig. 1.8) [44].

**Exploration phase**, the customers write out the story cards that they wish to be included in first release. Each story card describes a feature to be included into the program. At the same time the project team familiarize themselves with the tools, technology and practices they will be using in the project. The technology to be used will be tested and architecture possibilities for the system are explored by building a prototype of the system. The exploration phase takes time between weeks to a few months.

**The Planning phase** sets the priority order for the stories and agreement of the contents of the first small release is made. Programmers estimate the effort required by each story and schedule is then agreed upon. The time span of schedule of the first release doesn’t take more than two months. Planning phase itself takes about few days.
Fig 1.8 XP Process

- Exploration Phase
- Planning Phase
- Iteration to release phase
- Productionizing Phase
- Maintenance Phase & Death

- Stories for next Iteration
- Effort Estimation
- Priorities

- Analysis
- Design
- Planning (testing)
- Testing

- Pair Programming

- Feedback
- Continuous Integration

- Test
- Collective Codebase

- Final Release
- Small Release
- Updated Release phase

- Regular Updates

- Continuous Review
The Iteration to release phase includes several iterations of the system before the first release. The schedule set in the planning stage is broken down to a number of iterations that will each take one to two weeks to implement. The first iteration creates a system with the architecture of the whole system. This is achieved by selecting the stories that will enforce building the structure for the whole system. The customer decides the stories to be selected for every iteration. The functional tests created by the customer are run at the end of every iteration. At the end of last iteration the system is ready for production.

The Productionizing phase requires extra testing and checking of the performance of the system before the system can be released to the customer. At this phase, new changes may still be found and the decision has to be made if they are included in the current release. During this phase, the iterations may need to be quickened from 3 weeks to 1 week. The postponed ideas and suggestions are documented for later implementations during maintenance phase. After the first release is productionized for customer use, the XP project must both keep the system in the production running while also producing new iterations.

Maintenance phase has been introduced in order to do above. Maintenance phase may require incorporating new people into the team and changing the team structure.

The Death phase is near when the customer does no longer have any stories to be implemented. This requires that the system satisfies customer needs also in
other respects. This is the time in XP process when necessary documentation of the system is finally written as no more changes to the architecture, design or code are made. Death may also occur if the system is not delivering the desired outcomes, or if it becomes too expensive for further development.

The differences between XP and Scrum are often quite subtle, but they are important. Scrum and XP are often used together because their practices are more or less disjoint. Both of these approaches are often billed as a complete set of practices that are supposed to be used without modification, at least until the users become skilled enough to make appropriate changes. However, when used together, a few differences have to be resolved, including recommended iteration length, specific planning details including release planning and stories or backlog items. There are six main differences [45] between Scrum and XP which are mentioned below:

- Scrum focuses on project management while XP focuses on developer practices.
- Scrum has the role of a Scrum Master, a person who organizes the daily Scrum meeting, runs interference with management, and helps break down barriers. Outside consultants are not generally considered Scrum Masters, they assist Scrum Masters. This is different from XP coach who is a
consultant that comes from time-to-time to give advice, and a coach who is involved in the day-to-day activities of the team one or two weeks long.

- Scrum teams do not allow changes into their sprints. Once the sprint planning meeting is completed and a commitment made for delivering a set of product backlog items, that set of items remains unchanged through the end of the sprint. XP teams are much more amenable to change within their iterations. As long as the team hasn’t started work on a particular feature, a new feature of equivalent size can be swapped into the XP team’s iteration in exchange for the not started feature.

- Extreme Programming teams work in a strict priority order. Features to be developed are prioritized by the customer (Scrum’s Product Owner) and the team is required to work on them in that order. By contrast, the Scrum product owner prioritizes the product backlog but the team determines the sequence in which they will develop the backlog items.

- Scrum doesn’t prescribe any engineering practices; XP does. XP engineering practices, like test-driven development, the focus on automated testing, pair programming, simple design, refactoring, and so on.

These are small and often subtle differences between Scrum and XP. However, they can have a profound impact on the team. The XP practices are
wonderful but they work best and teams commit to them the most stridently if they discover them themselves rather than having them mandated.

1.6.3 Dynamic systems development method (DSDM)

DSDM has gradually become the number one framework for rapid application development (RAD) in the UK. DSDM is a non-profit and nonproprietary framework for RAD development, maintained by DSDM consortium.

The DSDM philosophy is that any project must be aligned to clearly defined strategic goals and focus upon early delivery of real benefits to business. DSDM is vendor independent, covers the entire lifecycle of a project and provides best practical guidance for on time, in budget delivery of projects, with proven scalability to address projects of all sizes and for any business sector [46]. DSDM provides a framework of controls for RAD, supplemented with guidance on how to efficiently use these controls. The fundamental idea behind DSDM is that instead of fixing the amount of functionality in a product, and then adjusting time and resources to reach that functionality, it is preferred to fix time and resources and then adjust the amount of functionality accordingly [47]. It is developed by DSDM consortium. DSDM, in 1994, grew to provide an industry standard project delivery framework for what was referred to as Rapid Application Development (RAD) at that time. While RAD was extremely popular in the early 1990’s, the RAD
approach to software delivery evolved in a fairly unstructured manner. As a result, the DSDM consortium was created. DSDM is based on nine key principles that primarily revolve around business needs/value, active user involvement, empowered teams, frequent delivery, integrated testing, and stakeholder collaboration. DSDM specifically calls out “fitness for business purpose” as the primary criteria for delivery and acceptance of a system, focusing on the useful 80% of the system that can be deployed in 20% of the time [48]. DSDM is designed for small teams, but we can scale it up. Requirements are planned and delivered in short, fixed-length time-boxes, also referred to as iterations, and requirements.

Fig 1.9 DSDM
Five phases of DSDM are Feasibility, Business study, Functional model iteration, Design and build iteration and Implementation (Fig 1.9).

The first two phases are sequential and done only once. The last three phases, during which the actual development work is done, are iterative and incremental. DSDM approaches iterations as timeboxes. A timebox lasts for a predefined period of time, and the iteration has to end within the timebox. All critical work must be completed in a DSDM project. It is also important that not every requirement in a project or time-box is considered critical. Within each time-box, less critical items are included so that if necessary, they can be removed to keep from impacting higher priority requirements on the schedule. Explanations for the above phases are [34]:

**Feasibility study phase** is where the suitability of DSDM for a given project is assessed. Judging by the type of project and most of all, organizational and people issues, the decision is made, whether to use DSDM or not. Two work products are prepared- a feasibility report and an outline plan for development. Optionally, a fast prototype can also be made if the business or technologies are not known well enough to be able to decide whether to proceed to the next phase or not. The feasibility study phase is not expected to take more than a few weeks.
The business study is the phase where essential characteristics of the business and technology are analyzed. The recommended approach is to organize workshops, where a sufficient number of customer’s experts are gathered to be able to consider all relevant facets of the system to be able to agree on the development priorities.

The functional model iteration phase is the first iterative and incremental phase. In every iteration the contents and approach for iteration are planned, the iteration goes through and results analyzed for further iterations. Both analysis and coding are done, prototypes are built and experiences gained from them are used in improving the analysis models. The prototypes are not to be entirely discarded. A functional model is produced as an output containing the prototype code and analysis models. Testing is also continuing and essential part of this phase. There are four outputs in the phase. Prioritized function is the prioritized list of the function that is delivered at the need of the iteration. Functional prototyping review documents collect the user comments about the current increment, working as input for subsequent iterations. Nonfunctional requirements are listed, mainly to be dealt within the next phase. Risk analysis of further development is important document in the function model iteration phase, because from next phase onwards, encountered problem will be more difficult to address.

The design and built iteration is where the system is mainly built. The output is a tested system that fulfills at least the minimum agreed set of
requirements. Further development is based on user’s comment. The final implementation phase is where the system is transferred from development environment into actual production environment. Trainings are given to users and the system is handled over to them.

For DSDM projects are prioritized using MoSCoW rules [49]

M- Must have requirements
S - Should have if at all possible
C- Could have but not critical
W - Won’t have this time, but potentially later

The DSDM project framework is independent of, and can be implemented in conjunction with, other iterative methodologies such as Extreme Programming and the Rational Unified Process.

1.6.4 Feature driven development (FDD)

FDD was originally developed and articulated by Jeff De Luca, with contributions by M.A. Rajashima, Lim Bak Wee, Paul Szego, Jon Kern and Stephen Palmer. The first incarnations of FDD occurred as a result of collaboration between De Luca and OOD thought leader Peter Coad. FDD is a model-driven, short-iteration process. It begins with establishing an overall model shape. Then it continues with a series of two-week “design by feature, build by feature” iterations
FDD expects requirements to be well captured and understood. FDD recommends specific programmer practices such as "Regular Builds" and "Component/Class Ownership". FDD's proponents claim that it scales more straightforwardly than other approaches, and is better suited to larger teams. Unlike other agile approaches, FDD describes specific, very short phases of work which are to be accomplished separately. These include Domain Walkthrough, Design, Design Inspection, Code, Code Inspection, and Promote to Build. It does not cover the entire software development process, but focus on the design and building phase.

1.6.4.1 Leadership members of FDD

FDD consists of leadership members such as:

- **Project manager**
  Project manager is administrative lead, responsible for securing funding and other resources and for reporting progress. This role is not like a Scrum master, but has some similarities. Chief Architect is responsible for system design, but more by providing guidance than direction. She/he is similar to the technical coordinator of DSDM.

- **Development manager**
  Development manager is responsible for day-to-day development activities.

- **Chief programmers**
Provide technical leadership to small teams. Sort of like mini-chief architects for sub-portions of a large project.

- **Class owners**
  
  Each class is owned by a single developer, who makes all changes to it.

- **Domain experts**

  Every ecosystem has to have users.

### 1.6.4.2 Phases of FDD

FDD consists of 5 sequential processes (Fig. 1.10) which are summarized in the following ways [51]:

![Fig 1.10 Feature-Driven Development](image)

Develop an overall model phase with the beginning of this phase domain experts are already aware of the scope, context and requirements of the system to be built. Documented requirements such as use cases or functional specifications are
likely to exist at this phase. After each walkthrough, a development team works in small group in order to produce object models for the domain area at hand. The development team then discusses and decides upon the appropriate object model for each domain areas.

**Build a feature list phase** the walkthrough object models and existing requirement documentation give a good basis for building a comprehensive features list for the system being developed.

**Plan by feature phase** it includes creation of a high-level plan, in which the feature sets are sequenced according to their priority and dependencies and assigned to chief programmers. Further, the classes identified in the process are assigned to individual developers, i.e. class owners.

**Design by feature and build by feature phase** a small group of features is selected from the feature sets, and feature teams needed for developing the selected features are formed by the class owners. The design by feature and build by feature processes are iterative producers, during which the selected features are produced. One iteration should take few days to a maximum of 2 weeks.

### 1.7 Lean software development

Lean is used when demand increases too folds and a single mistake could lead to business loss. Traditional business practices are often poorly suited to deal with these expectations, and management theorists increasingly restless in their
pursuit of new understanding. For sustainability in market some of the things listed below are needed:

1. Fast delivery i.e. deadline meeting.
2. Easy management.
3. Highly risk prone.
5. Maintain financial strength

Lean offers all the above things to survive in today’s world. Lean software development is an iterative methodology originally developed by Mary and Tom Poppendieck. Lean software engineering received much attention primarily from industry after the publication of a book on lean software development by Poppendieck and Poppendieck. The Poppendiecks have practical experience from lean manufacturing and product development. When Mary Poppendick got to know about the Waterfall model in software development, she recognized that the software community could benefit from the ideas of flexible production processes. This was the motivation for them to look into how the Lean principles and practices from product development and manufacturing could be used in the case of software development. The main idea behind Lean is to focus all development effort on value adding activities from a customer’s perspective and to systematically analyze software processes to identify the waste and then remove it
Lean software development helps software organizations to optimize their processes and production methods in order to deliver their products to the market much faster and with better quality. Lean puts main focus on people and communication. People who produce the software are respected and they communicate efficiently. It is more likely that they will deliver good product and the final customer will be satisfied. Lean software development subsequently gave birth to agile software development methods and its main branches are Scrum and Crystal Clear [54]. Lean says that different wastes that are considered as not contributing to the value of the customer should be removed. Lean eliminates waste through such practices as selecting only the truly valuable features for a system, prioritizing those selected, and delivering them in small batches. It emphasizes the speed and efficiency of development workflow, and relies on rapid and reliable feedback between programmers and customers. Lean also strongly recommends that automated unit tests be written at the same time the code is written. Lean software development is not a management or development methodology, but it offers principles that are applicable in any environment to improve software development [49].

The main principles [55] [56] of Lean include:

1. Eliminating waste
2. Inventory is waste
3. Maximize flow
4. Pull from demand and decide as late as possible

5. Empowering the Team

The first rule of Lean Manufacturing and TQM is elimination of waste. That is, eliminate anything which does not add value to the final product. Documents, diagrams and models produced as part of the software development must be minimized because once a working system is delivered the user may care little about these deliverables.

The second rule of Lean Manufacturing and TQM is that inventory is waste. Inventory consumes resources, slows down response time and becomes obsolete. The inventory of software development is documentation; excess documentation creates a waste of time in producing and reviewing the documents. Rather than having a 100 page detailed specification, write a 10 page set of rules and guidelines. This is what agile methodologies rigorously maintain, and documentation should be kept to minimal.

The third rule of Lean Manufacturing and TQM is to maximize flow. Rather than taking months to show the customer the final product, use an iterative development where small but complete portions of a system are designed and delivered throughout the development cycle. Similar to agile methods this technique allows the customer to have a better idea of how the software works.
The fourth rule of Lean Manufacturing and TQM is pull from demand and deciding as late as possible. Software development practices which keep requirements flexible and as close to system delivery as possible can provide a significant competitive advantage in changing requirements. Similarly, agile methodologies are designed to respond to change, not predict it, and have the ability to make decisions as late as possible.

The fifth rule of Lean Manufacturing and TQM is to empower workers, to provide both the tools and the authority for people other than managers to make decisions. This is one of the problems with heavyweight documentation that it does not attempt to make all of the decisions for developers. However agile methodologies give developers guidance as well as freedom to make the detailed design and programming decisions.

Following are the differences (Table 1.1) between Lean and agile on the basis of customer and development speed [53].
<table>
<thead>
<tr>
<th>Aspect</th>
<th>Lean</th>
<th>Agile</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customer</td>
<td>Create value for the customer and thus only focus on value-adding activities.</td>
<td>Have a working product that fulfills the customer’s need.</td>
</tr>
<tr>
<td>Development speed</td>
<td>Rapid value creation and short cycle times.</td>
<td>Continuous delivery of working software.</td>
</tr>
<tr>
<td>Teams</td>
<td>Self-organizing teams.</td>
<td>Informal team organization</td>
</tr>
<tr>
<td>Focus</td>
<td>Lean development emphasizes the traditional lean concepts of value and flow.</td>
<td>Agile software development focuses on emergence.</td>
</tr>
</tbody>
</table>

Table 1.1 Difference between Lean and Agile Software Development

Delivering fewer requirements or features more frequently avoids that requirements from becoming obsolete, and allows for much earlier feedback from testing and customers. Regarding the question whether Lean development is agile and agile development is Lean, following arguments can be provided: Lean is Agile as it includes all the principles of agile. However, agile is not Lean as it does not emphasize the E2E flow. The E2E flow includes principles that emphasize the focus on the overall flow of value (i.e. from the very beginning when a need for a feature enters the organization till it is delivered).

Lean is very similar to Scrum of agile. It is similar in the sense that in this focus is on features instead of group of features (Fig. 1.11 and Fig. 1.12). Lean has
one step further than Scrum. So, in Lean feature is deployed that we want to create but in Scrum group of features is deployed
1.7.1. Kanban

Kanban is the term David Anderson coined with respect to an agile development approach to driving change based on lean principles. Kanban is a system to control the logistical chain from production point of view, and is not an inventory control system. Kanban was developed by Taiichi Ohno, at Toyota, to find a system to improve and maintain a high level of production. Kanban is one method through which JIT is achieved [57].

It’s no process framework at all, but a model for introducing change through incremental improvements. Kanban principles can apply to any process that is already running. In Kanban, work is organized on a Kanban Board. There are states through which every work item passes through from left to right. For every column (state) on Kanban board there should be a “Work In Progress” Limit (WIP Limit) defined. The WIP limit tells how much work items are allowed to stay in a certain state. If the state is “full”, no new work can enter that state. In this way team will find out about bottlenecks in the progress simply by looking at the Kanban Board and is challenged to change the way of work to avoid bottlenecks in the future. In that way, the WIP limit acts as change agent in Kanban.

Scrum vs. Kanban

If organization is really stuck and needs a fundamental shift towards a more efficient process, Scrum seems to be more appropriate. If organization already
have working processes, which we want to, improve over time without shaking up the whole system, Kanban should be tool of choice [58].

1.8 Scope of proposed work

In past main methodology was heavyweight methodology which consists of heavy documentation, comprehensive planning and extensive design. New methodologies such as lightweight methodologies emphasis is on working software rather than documentation and responding to change over following a specific plan. According to a survey conducted on heavyweight methods by IBM it was showed that software developed costs more than projected, it took longer to complete than what was predicted, and most of projects had to be substantially redesigned. Due to all these problems main objective of this thesis is to study about lightweight methods called agile methodologies and difference between different agile methodologies, so that according to their characteristics a person can apply it to her/his organization. The thesis also aims to produce comparison of agile with heavyweight methodologies in Indian organizations. For main results Hypothesis are designed and proved by using one way ANOVA method. Three hypotheses that are designed for Indian organizations are:

**Hypothesis I**

Production gets increased on using different methodologies of agile instead of heavyweight methods.
Hypothesis II
Quality gets increased on using different methodologies of agile instead of heavyweight methods.

Hypothesis III
Cost gets reduced on using different methodologies of agile instead of heavyweight methods.

1.9 Organization of thesis

The structure of this thesis is as follows:

Chapter 2

In this chapter a brief study on the related work was carried out in area of agile. The research papers, books, magazines, online links and journals that helped to start this research work have been discussed. This chapter discusses how software developers moved towards agile following a lot of steps such as main process models of software development. Why they moved towards iterative change driven models of software development and then fundamentals of agile software development. Current status of agile software development has also been discussed.

Chapter 3

This chapter contains methodology and formulation of problem. It discusses data collection and analysis. Two Hypotheses are designed to find out either there is
increase in production and quality or not. Moreover one Hypothesis is designed to find out that there is decrease in cost or not.

Chapter 4

This chapter is about results which resulted from questionnaire, interview and online surveys. Findings are discussed in this chapter using graphs.

Chapter 5

This chapter is about limitations of agile methods even though agile is good for so many reasons.

Chapter 6

This chapter is about conclusions of the overall thesis and directions for future research.

Finally, the thesis presents some of the publications that resulted out from the present research work.