Chapter 4

EXISTING AGILE METHODOLOGIES

4.1 Introduction
In this chapter some of the common agile methods along with their working are introduced. Then comparisons among traditional and agile approaches as well as comparisons among various agile methodologies are described here. After comparing various agile methodologies, the best one is selected.

4.2 Existing Agile Methodologies
There are number of agile methodologies. These are as follows.
- Extreme Programming
- Scrum
- Lean Development
- Feature Development
- DSDM

4.2.1 Extreme Programming
Extreme Programming was introduced by Kent Beck in 2000[19, 28]. This is an agile software development method. This methodology arose from the problems caused by the long development cycles of the traditional development models. It started as “simply an opportunity to get the job done” with practices that had been found effective in software development in the earlier decades. After many successful trials in practice the XP methodology was “theorized” on the key principles and practices. It can be described as a set of rules rather than methodology of working. It is a set of rules and processes to develop software; it is frame under the agile methodologies. XP is intended to improve software quality and responsiveness to the change of customer’s requirements.
It is a deliverable and disciplined approach to software management which takes a code-centric view of the activity. It is a package of several practices and ideas which
are not new. The combination and packaging of all of these is however new. XP is targeted at small co-located teams developing new critical products.

The main objective of this approach is to deliver what the customer needs at the time it is needed. The waterfall process does not perform well, because it cannot adapt to changes. Usually the customers change their minds because often they do not know in advance what they want. This determines the need for changes later during the development. Waterfall process entails a long development cycle, developing software following a rigid schema where the collection of requirements is done as the first step of the cycle. Consequently, it is very difficult to change the expected software features when the analysis phase is completed. The conventional Waterfall software process is overcame by XP Process. Rather than planning, analyzing, and designing for a long time, XP exploits the reduction in the cost of changing software to do all of these activities a little at a time, throughout software development. XP is known as a methodology [27] that “reduces bugs” in software, which is a great selling point.

XP is a new way of building the right software, and building it fast, to a high quality and explain how it has been tried on real projects that proved it work with a small development team.

4.2.1.1 Rules of XP

1. Only write code for the current task. Never try to use the actual task time for developing code of a different task, even if the task are very related.

2. Re-factoring
   Try to get rid of the duplication of code, make complex code simpler.

3. Integrate often
   Developers should be integrating and committing code into the repository every few hours, whenever possible. In any case never hold onto changes for more than a day. Continuous integration often avoids diverging or fragmented development efforts, where developers are not communicating with each other about what can be re-used, or what could be shared.
4.2.1.2 Documents and Artifacts

XP depends on documentation via oral communication, code itself and knowledge transfer rather than written documents and artifacts. However oral communication may be suited for small groups but not recommended for large as well as high risk involving systems [17]. In such cases following tools may need to be managed.

- **User story cards**: These cards are paper index cards which contain brief descriptions. These cards are not a full requirement statement rather a commitment between developer and the customer for further conversation. During this conversation both parties will come to an oral understanding of what is needed for the requirement to be fulfilled. Priorities of the customers as well as resource estimate by the developer are also added to these cards but resource estimate for a user story must not exceed the iteration duration.

- **Task List**: This list contains a listing of tasks for the user stories that are to be completed for iteration and duration of this list may be one half to three days in duration.

- **CRC Cards**: CRC stands for Class Responsibility Collaboration. It is paper index card on which one records the responsibilities and collaborators of classes which can serve as a basis for software design. The classes, responsibilities and collaborators identify a design involving multiple developers.

- **Customer acceptance Tests**: Textual descriptions and automated test cases are developed by the customer. The development team demonstrates the completion of a user story and the validation of customer requirements by passing these test cases.

- **Visible wall graphs**: To foster communication and accountability, progress graphs are usually posted in team work area. How many stories are completed and how many acceptance test cases are passing are involved in these progress graphs.
4.2.1.3 Phases of XP

The life cycle of XP can be broken down into phases.

1. **Exploration**
   
   In this phase the requirements of the user about the system are written on story cards. Sufficient story cards are produced to satisfy the customer that the material will make a good first release. At the similar time it is also decided by the development team that what technology to be used and which architecture to be used etc. It means they get familiar with the technology to be used. It can have duration from few weeks to few months. This is a phase where the system gets its form and the development team uses the technology.

2. **Planning**
   
   User stories will be arranged in a priority order in a planning phase and the contents of the first small release is agreed. This phase takes a couple of days and first release is taken not more than two months.

3. **Iteration**
   
   To create the first release, this phase consists of the several iterations of the system. The first iteration creates the basic architecture of the system by implementing the
user stories. Functional tests created by the customer are run at the end of each iteration cycle.

4. **Productionizing**

   Additional testing and checking is conducted in the productionizing phase before releasing the system to the customer. At this stage new changes can be found and a decision is taken whether it should be included or not. The implementation for the changes may need to be shortened from three weeks to one week. Changes which are postponed are documented for late implementation.

5. **Maintenance**

   In the maintenance phase the project team maintains the running system and at the same time works on new product increments. This phase may require changing the team structure by incorporating new people into project.

6. **Death**

   When the customer is satisfied with the project and does not have any stories to be implemented, the project is reached at the death stage. Necessary documentation of the system is finally written as no changing is required to the architecture. Death may also occur, if the product is terminated for some reason like the project is too expensive for further development or it fails to deliver the desired outcome.

### 4.2.1.4 There are different actors who take part in the XP process

1. **Manager** owns the team and its problems. Manager is the person who owns the team, obtain resources, and manage people and problems and interfaces with external groups.

2. **Tracker** who is responsible for tracking time estimations, progress and probable risks. He regularly collects user story and acceptance test case progress from the developers to create the visible wall graphs.

3. **Coach** is typical a programmer and not a manager. He or she teaches team members about the XP Process as necessary, intervenes in case of issues; monitors whether the XP process is being followed.

4. **Customer** who is responsible for writing story cards, giving priorities to these story cards and writing and performing functional tests. A common
misconception is that the role of the customer must be played by one individual from the customer organization. Conversely, a group of customers can be involved or a customer representative can be chosen from within the development organization but it should be external from the development team.

5. **Programmer**, who estimates the required efforts for each story card, writes tests and code that fulfills these tests.

6. **Tester** may be a programmer also and he helps the customers to write and develop tests.

4.2.1.5 Principles of XP

1. **Planning Game**
   
   The planning game in XP is XP’s response to the problem that you cannot know everything in advance, especially in software engineering. The main idea in this game is to form a rough plan of the next iteration quickly, and then refine it as the iteration moves on. The planning game is a collaborative game of customers and developers where both work very closely. The developers contribute by estimating how much implementing a software feature will cost, and the business people must decide which features will be included to the iteration and what are their priorities.

2. **Small Releases**

   In this approach emphasis is given that rather than once or twice a year new version of the software should be released in iterations of one or two months. Short or frequent iterations provide early value to the customers and enable him to quickly verify the developed software. It reduces customer risk and providing valuable feedback to the development team.

3. **Simple Design**

   XP Methodology favours a simple design. The idea is to develop a simple solution that could possible work for the current feature. It is useless to design a system as it is as at the end of the project because the future is uncertain. When the design solution is not good enough, it is re-factored to meet the new requirements. The idea of designing through re-factoring keeps the code clear. It
allows the developers to design only what is required to implement the next feature. It does not contain any functionality that is not currently needed.

4. **Testing**
   
   Testing is another important aspect of Agile Methodologies. In waterfall model, testing takes place at the end of the development cycle whereas agile methods advocate test-driven development which states that tests must be written before the code itself and that a failing test must be written before a defect is fixed. Every programme must be featured with these tests which provide rapid feedback on the effects of a change.

5. **Pair Programming**
   
   Pair programming means every line of code is written in pairs, with two developers working on one computer. The one is holding the keyboard as well as mouse, the other one is looking at the general view, thinking of the possible simplifications and consequences. Pair Programming provides various benefits. At least two developers are familiar with every part of the system and when working as a pair, they motivate each other to better development practices. The developer’s pair spreads the knowledge of the system inside the team.

6. **Re-factoring**
   
   The process of changing the internal structure of the code without changing its functionality is called Re-factoring. This is applied for various reasons like improve communications, remove duplication, add flexibility and to simplify the design. Developers look at the code and think of ways for changing it when they want to add a new feature. After adding the feature developers look at the code and think of ways to simplify the resulting design.

7. **Metaphor**
   
   It is a shared story between customer and the developers and creates a common vocabulary as well as guides to the developers when developing the software. The XP Metaphor replaces the architecture in traditional software.
4.2.1.6 XP is not suited in every environment. The areas covered by XP are as follows:-

1. XP works for those projects which are dynamic in nature as well as not clear at the time of beginning.
2. XP is not well suited for large teams. Teams with 2 to 12 members are best for XP approach.
3. If there is a strong co-operation between developers and customers, then XP is very good. If this is missing then XP approach does not produce good results.
4. XP works well for those projects also that cover high risk.

4.2.2 Scrum

This methodology was developed in 1996 by Jeff Sutherland and Ken Schwaber and its primary goal is to deliver software that, after each and every iteration, provides the highest business value. Scrum is an agile software development process designed to add energy, focus, clarity, and transparency to project teams developing software systems. It is based on a 30-day iteration called a "Sprint." Technically Sprints can be either two or four weeks, but the generally accepted default is usually four weeks. Scrum is agile software project management method. It will not define the way how software is developed, what documents are to be produced, how requirements should be gathered rather it is a guide how an agile implementation team should be managed. Scrum is not an independent stand alone development methodology; rather it is a wrapper for existing engineering processes. It manages and controls development work incrementally, improves communications, removes the problems to development with all in environment with rapidly changing requirements [19].

4.2.2.1 Definition of Scrum

“Scrum… is a framework with in which you can employ various processes and techniques… with in which complex products can be developed” – Ken Schwaber, Scrum Guide, May 2009.
4.2.2.2 Characteristics of Scrum Methodology

- Planning and closure phases consist of defined processes, where all processes, inputs and outputs are well defined. The knowledge of how to do these processes is explicit.
- The Sprint phase is an experiential process and many of the processes in this phase are unidentified or uncontrolled. It is treated as a black box that requires external controls including risk management, are put on each iteration of the Sprint phase to avoid confusion while maximizing flexibility.
- Sprints are nonlinear and flexible. Where available, explicit process knowledge is used; otherwise tacit knowledge and trial and error is used to build process knowledge. Sprints are used to develop the final product.

4.2.2.3 Phases of Scrum

**Pre-Game**

- Planning and High-Level Design

**Mid-Game**

- Sprint Plan
- Develop
- Adjust
- Wrap
- Review

**Post-Game**

- Close

Figure 4.2 Phases of Scrum

- Pre-game phase

All the requirements and system features are collected into a Product backlog in this phase. After that requirements are prioritized and what efforts are needed for their implementation is also estimated. The product backlog is kept up-to-date and at each iteration, it is reviewed by the Scrum team. There are no comments by scrum how and when the product backlog came into existence, but only that it must exist before implementation when it needs to be updated. A system architecture which is based on the product backlog is also created in this phase and as a result of new product
backlog if any problems arise related to the architecture, the problems and their solutions are included in the product backlog.

- **Implementation phase**
  The aim of scrum is to control the uncertain and changing variables of the development such as time frame, quality, requirements, resources, implementation technologies and tools, and even development rather than specifying them at the beginning. Scrum controls the development in agile iterations called sprints. Sprints are iterative development cycles during which the development team implements the selected items from the product backlog. Each sprint includes the traditional phases of software development like requirements, analysis, design, evolution and delivery. The architecture and the design of the system evolve during sprint development. A sprint may have duration from one to four weeks. Again, Scrum does not comment on how the development team works internally.

- **Post-game phase**
  All items in the product backlog are implemented in the post-game phase. Documentation, Integration and System testing are also included in this phase. The system is now ready to be released [22].

### 4.2.2.4 Advantages of Scrum Methodology

1. A Quality Product in a scheduled time can be delivered very easily.
2. It becomes easier to cope with the changes due to short sprints and constant feedback.
3. Through the daily meetings, issues are identified well in advance and hence they can be resolved in timely.
4. It is iterative in nature and requires constant feedback from the user.
5. It insists on frequent updating of the progress in work through regular meetings, so there is a clear visibility of the project development.
6. It helps the companies in saving time and money.
7. In scrum daily meetings make it possible to measure individual productivity. It motivates to improvement in the productivity of the each team member.
4.2.2.5 Disadvantages of Scrum Methodology

1. During a development work if any member leaves it, it puts a huge negative effect on the project development.

2. It is not fit for large projects. It is good for small, fast moving projects as it works well only with small team.

3. When the scrum master trust the team, this methodology works well. If he put too much strict control over the team members, it can be frustrating for them and become the reason for failure of product.

4. Experienced staff is necessary for the success of this methodology. If the team consists of beginners, the project can not be completed in time.

5. Tasks should be well defined. If the task is not well defined, time and estimating project cost will not be accurate. As a result task can be spread over several sprints.

6. It is not best suited for products where the focus is on usability. It fails to address usability needs of the user, because product owners keep their focus mainly on business issues and forget about usability.

4.2.2.6 Scrum Roles

There are 3 Primary roles in scrum methodology. These are:

1. Product Owner

2. Team Members

3. Scrum Master

1. **Product Owner**

   It is person who is responsible for taking the inputs from the customers, team members, end users and stakeholders and translating them into product vision.

2. **Team Members**

   The members of the team build the product that the customer is going to consume. It can be software, website or whatever it may be. The size of ideal team is
typically five to ten people. It can be maximum up to 15 and small up to 3. To deliver the finished product all expertise should be included in team members. Like in software engineering project team should include Programmers, testers, interface designers, marketers, researchers etc. Team members not only build the project but also give new ideas regarding the project to the Product owner.

3. **Scrum Master**

To become successful whatever is necessary by the team is provided by the Scrum Master. He/she serves the team by removing blocks to the team’s success, facilitating meetings and supporting the practice of Scrum. In some cases scrum master is fully dedicated to the role of scrum master where as in some cases scrum master can be a member of team. Product Owner and the Scrum Master should not be the same individual. Unlike Product Owner, Scrum master does not tell what to do or assign the work to people rather they facilitate the process to enable the team to organize and manage itself.

**4.2.2.7 Practices**

**Product Backlog**

It is kept in a spreadsheet like format. It contains the prioritized requirements that need to be developed into the system and the defects that need to be fixed during the release. For each requirement, the Product backlog contains a unique identifier for the requirement, the category, the status, the priority and the estimate for the feature. The product backlog is always updated when new requirements are discovered or current ones are refined and it is never completed during the project. When all items in the backlog have been finished, the project is complete. It is the responsibility of the product owner to maintain the product backlog.

**Sprint Burn-down Chart**

A tool for maintaining the product backlog item status is the burn-down chart, which is a table that can be used for tracking the items. It shows the amount of work remaining for each item across iterations.
Sprint Planning Meeting
A two-part sprint planning meeting is held before each sprint. In the first part of this meeting the goals for the next sprint are decided and prioritized. The most important features from the product backlog are selected by the product owner and the team then tells how many of these they can implement during the sprint. In the second part, the team lays down preliminary plans for the sprint by creating tasks to the sprint backlog.

Sprint
A sprint is originally 30-day iteration. Each sprint delivers valuable functionality to the developed system. During a sprint with the aid of Scrum Master, the Scrum team selects the appropriate methods for reaching that goal.

Sprint Backlog
The starting point for each sprint is sprint backlog. In the beginning, a set of requirements is taken from the product backlog and expanded into the sprint backlog. During the sprint these are then refined, but new requirements are not added to the list. New requirements can only be added to the product backlog.

Daily Scrum Meeting
Scrum introduces daily Scrum meetings to keep every project member up-to-date with the current situation. These meetings follow a very well-defined form and are short status meetings. In each meeting, three questions are asked from each developer: what has the developer done since the last meeting, what does the developer plan to do before the next meeting, and are there any obstacles in the developer’s way. To keep the meeting focused but still allow everyone to get a clear picture of how the project is proceeding is the purpose of these formal questions. Any problems identified are not discussed in the meeting, but another meeting will be scheduled for the team members who should discuss the problem. The meetings have also been useful as other developers have often battled with the same problems and could share their experiences after the daily meeting.
Sprint Review Meeting

A sprint review meeting is held after each iteration. In this meeting the produced software is introduced to the rest of the project stakeholders. The contents of the next sprint are also discussed in this meeting.

Effort Estimation

In Scrum, effort estimation is an iterative process. At first, initial estimates are filled to the product backlog. After more accurate estimations are formed, the new estimates are added as new columns to the product-backlog chart. By doing this the whole estimate history can be tracked from the product-backlog.

4.2.2.8 Process

- A Sprint Planning meeting is held with the development team, management and the product owner. The product owner is a representative of the customer or a contingent of customers. The product owner creates and prioritizes the product backlog. In the planning meeting, the product owner chooses which features are to be included in the next 30-day increment usually driven by highest business value and risk. A sprint goal is established also to keep the scrum team focused on the big picture, rather than just on the chosen features. Once the set of features has been identified, no re-prioritization takes place during the ensuing 30-day sprint in which features are designed, implemented and tested.

- Code is integrated and regression tested daily during a sprint.

- 15-Minutes Scrum Meetings are held daily. The meeting is held in a room with a whiteboard so that tasks and blocks can be written down. Social promises are made in the meeting. These meetings can be well managed with the smaller groups. If meeting is run with too many people, these meetings can become unmanageable. For successful meetings it is recommended that each team has a maximum of seven members. For larger teams, the team subdivides into smaller groups, each having its own scrum meeting. One representative from each of the smaller groups attends “Scrum of Scrums” meeting. The representative answers
the scrum questions, highlighting the activities of the team and in this way essential information are passed between sub-teams.

- At the end of a Sprint, a sprint review takes place to review progress, demonstrate features to the customer, management and review the project from a technical perspective. The latest version of the product is demonstrated in which the functions, design, strength and weaknesses are shared with the product owner.
- The contents of the next sprint are also discussed in this meeting and the cycle continues.

4.2.3 Lean Development

In 1989, Professor James Womack and consultant Daniel Jones published ‘Lean Thinking’ a survey of the lean manufacturing techniques that helped create the “Japanese miracle” of the late 1980s and early 1990s. They chronicled the ideas of lean manufacturing, with their focus on eliminating waste, creating a smooth “flow” of work on the factory floor, and expecting workers to contribute high skill levels and an ownership mentality.

Lean development is important not just for its conformance to the ideals of agile development but because the underlying philosophies of lean manufacturing have been accepted by business leaders worldwide, and so come with existing acceptance. This makes introduction of agile methods in a lean framework more easily accepted and presents a strategic framework that executives are likely to accept with less resistance[16].

The fundamental principle is to create change tolerant software with one–third the development hours, one–third the human effort, one–third the tools and methods and one–third the effort to adapt to a new market environment. 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, and when they ask for something it must be given to them so fast they don’t have time to change their minds. There is no team size specification because LD is more of a software development management philosophy than a methodology. It is used in any software development project where there is need for radical change.
4.2.3.1 Phases of Lean Development

Lean Software development process is composed of Four Phases:

I. Preparation
II. Planning
III. Implementation
IV. Assessment

I. In preparation phase an initial backlog of prioritized desirable stories is assembled at the beginning of development effort. Backlog items are usually features in terms of business goals since the Lean approach is to delay detailed analysis until the last responsible moment.

II. In the Planning Phase planning meeting is held at the beginning of iteration. The whole team makes estimations how long the top priority stories from backlog will
take to develop, test, document and deploy. On the basis of these estimations and
team capacity they pick the amount of stories they will be able to implement
during the iteration. Team members decide and commit to iteration goal, which
describes the theme of the feature set they picked for iteration.

III. A team develops, tests, documents and prepares for deployment the feature set
they picked during implementation phase. 10-15 minute team meetings are held
daily to discuss what each team member has accomplished since the last meeting,
what they will be doing till the next meeting, what problems they have, and where
they need help. A story is not considered done until the team updates all
associated artefacts like user documentation, design documents and other
artefacts.

IV. A review meeting is held at the end of iteration to show for the customer how
much value was added to the product during the iteration. If changes are required,
feedback from customer is collected. After this iteration assessment, planning
meeting starts for the next iteration.

4.2.3.2 LD emphasizes four key success factors that clearly illustrate its
compatibility with other agile methods:

1. Create visible customer value rapidly.
2. Build change-tolerant software.
3. Create only necessary functionality and no more.
4. Adopt aggressiveness, stubbornness, and belief in meeting stretch goals.

4.2.3.3 Techniques and Tools

To prove the value of every artifact, value analysis process is used. Rapid delivery of
product is another technique that races with changing requirements. The faster you
deliver, the least the change in requirements.

4.2.3.4 Outputs

Artifacts can be released incrementally as the customer requests before the final product.
4.2.3.5 Lean Development Agile method focuses on twelve management strategies, as follows

1. The highest priority is customer satisfaction.
2. Always provide the best value for the money.
3. Success depends on active customer participation.
4. Every Lean Development project is a team effort.
5. Everything is changeable.
6. Domain is not the point, however solutions are.
7. Complete, do not construct.
8. An 80% solution today, instead of a 100% solution tomorrow.
9. Minimalism is essential.
11. Product growth is feature growth, not size growth.

4.2.4 Feature Driven Development

The Feature Driven Development method was originally conceived by Peter Coad and his colleagues as a practical process model for object oriented software engineering. Stephen palmer and John Felsing have extended and enhanced Coad’s work, describing an adaptive, agile process that can be applied to moderately sized and larger software projects.

It has some common characteristics with XP such as short iterations, customer representative on the team & using the notion of features which are comparable to the XP. Prioritization are encouraged in FDD and it is ensured that most valuable features are delivered. FDD feature short iteration cycles i.e. two weeks. This methodology recommends individual code ownership and clearly defined roles.
4.2.4.1 Roles played in FDD

1. Class owner

Individual who is responsible for designing, coding, testing and documenting new features in the classes that he is assigned to him is known as Class owner.

2. Chief Architect

It specializes in developing and responsible for the overall design of the system including running workshop design sessions with the team.

3. Chief Programmers

These are senior developers who are more productive than others in the team. The chief programmer provides the breadth of knowledge about the skeletal model to a feature team, participates in high level requirements analysis and design and aids the team in analysis, design and development of new features.

4. Domain experts

The persons who have deep knowledge of the business for which the product is being developed are known as domain experts. These experts may be users, clients, sponsors or business analysts etc.

5. Project Manager

The individual who is responsible for reporting progress, managing budgets and managing resources including people, equipment and space etc.

6. Development Manager

The person who is responsible for day to day development activities is known as development manager.
7. **Feature Teams**

A feature team dynamically forms to implement a feature and disbands when the feature has been implemented.

8. **Domain Manager**

It is the leader of the domain experts and resolves the arguments that may rise within the ranks of the experts.

9. **Release Manager**

By reviewing the progress reports from the chief programmers and by having short progress meetings with them it controls the progress of the process and reports to the project manager.

10. **Language Guru**

A team member who possesses a thorough knowledge of a certain programming language or technology and plays an important role when the development team has to work with some technology that is new to them.

11. **Build Engineer**

A Member who is responsible for setting up, maintaining and running the build process and manages the version control system and publishes documentation for it.

12. **System Administrator**

Configures, manages and troubleshoots the servers, workstations and development and testing environments that are needed in the project.
13. **Tester**

The person who verifies that the system will meet the requirements of the customer and may be working in an independent team or as a part of the project team.

14. **Technical Writer**

 Prepares the user documentation and may form an independent team or work as part of the project team.

![Figure 4.4 Phases of FDD](Image)
4.2.4.2 Process of FDD

1. **Develop an overall Model**

Class diagrams, Sequence diagrams and modeling alternatives are developed for each area of the problem domain. Model notes record information about the model’s shape and why some alternatives were selected and others rejected.

2. **Build a Features list**

The team identifies the features, groups them hierarchically, prioritizes them, and weights them. Smaller teams tackle specialized feature areas in subsequent iterations of this process, in many of those follow-up sessions domain members participate. The outcome is a detailed feature list.

3. **Planning**

Based on the information of the detailed feature list, Project Manager, Development Manager and the Chief Programmers established the milestones for the following two stages. Classes are assigned to class owners, an overall end date is determined as well as completion dates for the features. Features are bundled according to the technical reasons rather than business reasons.

4. **Design by Feature**

The corresponding class owners are grouped as a feature team for each feature. This team delivers detailed sequence diagrams and class or method updates. Domain experts interact with the team to refine the feature requirements. Designs are inspected and the results have to be approved by the chief programmer.

5. **Build by Feature**

Actual coding happens in this phase. Methods for the classes are defined, and unit testing is performed. After the unit tests pass, the classes are checked into the configuration management system. Classes and methods outlined by the design are implemented by the feature team.
4.2.4.3 Core Values of FDD

1. Putting in place a system for building systems is necessary for successful scaling of larger projects.
2. Putting together a simple, well-defined process that works best.
3. Ensuring process steps are logical.
4. Get rid of ‘Process pride’ as it keeps the real work from happening.
5. Good processes are moved to the background to allow team members to focus on results.
6. Short, iterative, feature-driven life cycles are considered the best.

4.2.4.4 Tools Used For Feature Driven Development

1. CASE Spec. CASE Spec is a commercial enterprise tool for Feature-Driven development.
2. TechExcel DevSuite. TechExcel DevSuite is a commercial suite of applications to enable Feature-Driven development.
3. FDD Project Manager Application. FDDPMA is a web-based effort that aims to facilitate iterative development by reducing FDD management overhead, producing graphical progress reports, and providing a workplace where all the FDD related documentation is collected.
4. FDD Tools. The FDD Tools project aims to produce an open source, cross-platform toolkit supporting the Feature Driven Development methodology.
5. FDD Viewer. FDD Viewer is a utility to display and print parking lots.

4.2.4.5 Documents

- Design Packages
  It consists of sequence diagrams and class diagrams and method design information.
• **Track by Feature**
  A chart which enlists the features to be built and the date of completion is known as track feature.

• **Feature lists**
  This document consists of a set of useful features in the eyes of clients that can be implemented in two or less weeks. If a feature would take more than two weeks to implement, it must be further decomposed.

• **Burn up Chart**
  A chart where date/time is displayed on X axis and increasing number of features that have been completed on Y axis is known as burn up chart. This chart indicates a positive slope over time as features are completed.

### 4.2.5 Dynamic Systems Development Method

The Dynamic Systems Development Method (DSDM) is a lightweight software methodology which has its origins in the U.K. Here time is fixed[19,22] for the life of a project, and resources are fixed as far as possible. This means that the requirements that will be satisfied are allowed to change. DSDM is focused on customer solutions and business value. The life cycle of DSDM consists of five phases. Out of five phases the first two are performed only once and the last three are iterative.

#### 4.2.5.1 Phases of DSDM

1. **Feasibility study**
   Here it is decided whether or not to use DSDM for the project. A technical feasibility study by building a prototype may also be there. A decision whether or not to start a project altogether is also taken. Feasibility study should be as short as possible.

2. **Business Study**
   To get a good understanding of the business processes that need to be automated is the goal of this stage. This stage again should be a short exercise where the outcome is a high level definition of the business process. Stake holder analysis is also
involved in this stage. Prototype activities and configuration management are also planned in this stage.

3. **Functional model iteration**

This phase focuses on refining high-level requirements of the business study. The outcome of this stage is an analysis model which is primarily geared to satisfy usability requirements. Testing is a vital component of this stage.

4. **Design and build iteration**

To refine the system to a sufficiently high standard so that it can be put into operation is the main goal of this stage. Intermediate work products are designed and functional prototypes. The outcome of this stage is a tested system and documentation as required.

5. **Implementation**

In DSDM, implementation denotes the transition from the development environment to the operational environment rather than coding. The system is handed over to the users who are then trained. Other work products include the user manual and a project review report [29, 36]. At the end of this stage, it is decided whether there is the need for iteration.

![Figure 4.5 Phases of DSDM](image)
4.2.5.2 Roles and Responsibilities

- **Developers and senior developers**
  In the development process developers and senior developers are involved.

- **Technical coordinator**
  The role of Technical co-coordinator is to define system architecture and ensures technical quality in the project.

- **Ambassador user**
  It brings the knowledge of users into the project and reports project progress to other users.

- **Visionary**
  It has accurate view of business objectives of the system and project.

- **Executive Sponsor**
  The financial authority and responsibility is given to the executive sponsor.

4.2.5.3 Scope

DSDM has been applied to small and large projects and mainly applicable to the development of user interface intensive systems and complex business applications, but may also be used for non–IT projects. Team size varies from two to six people but there may be many teams in a project.

4.2.5.4 Techniques and Tools

According to Stapleton [7]; business study workshop is a technique that brings together all stakeholders and elicits business requirements. DSDM does not specify tools but provides an ideal support environment, which spells out a need for requirement analysis tools, system prototyping tools, design tools, construction tools, testing tools, and reverse engineering tools. DSDM training is often recommended in order to effectively introduce it in an organization.
4.2.5.5 DSDM Principles

1. **Active user involvement is imperative.**
   DSDM is a user-centered method. The delays will occur if the real users are not closely involved in the development. The developers will make decisions without consulting the users, and the users may feel that the solution is forced by the developers and their management. In DSDM the users are actively involved in the development process.

2. **DSDM teams must be authorized to make decisions.**
   DSDM teams involve both developers and users, and they must be authorized to make decisions as the requirements refine and get possibly changed. Certain levels of functionality, usability etc. is acceptable without the frequent consultation of the higher management.

3. **The focus is on frequent delivery of products.**
   The work of a DSDM team focused on delivering products in decided period of time. These are not complete solutions, but just iterations towards the full product. This causes the team to select the best possible solution that can be achieved in the given timeframe. The time periods are kept short, so it is easy to decide which activities are needed to make the product.

4. **Fitness for business purpose is the essential criterion for acceptance of deliverables.**
   DSDM focuses to deliver the necessary functionality in the given time. The system can be more thoroughly engineered later on, if agreed so.

5. **Iterative and incremental development is necessary to converge on an accurate business solution.**
   DSDM allows systems to rise incrementally, so developers can fully use the feedback from the users. The partial solutions can also be delivered to satisfy immediate business needs. Rework is built into the DSDM process so it is easy to go back to a previous step, which speeds up the implementation.

6. **All changes during development are reversible.**
   Everything must be in a known state at all times to control the progress of all products (documents, software, test products, etc.). Backtracking is a feature of DSDM. The ability to reverse changes is limited to current increment.
7. **Requirements are base lined at a high level.**

The freezing and agreeing the purpose and scope of the system at a level allows examining the meaning of requirements. Later in the development process, the more detailed requirement base lines can be introduced, but the scope should not change significantly.

8. **Testing is integrated throughout the lifecycle.**

Testing is integrated to the development process and is not treated as a separate activity. During the development process, the system is reviewed and tested by developers and users incrementally to confirm that the development is going to the right direction. The business needs and priorities are validated in the early phases of DSDM and later testing focus shifts towards verifying that the system functions correctly and efficiently.

9. **A collaborative and co-operative approach between all stakeholders is essential.**

When the developers start their work, the low level requirements are not necessarily fixed. DSDM requires that the short term direction for the project must be decided quickly without recourse, so all stakeholders should have a collaborative and co-operative attitude.

4.3 **Comparison of Agile Methods with Traditional Methods**

Agile Methods are called Light Weight Methods and Traditional Methods are called Heavy Weight Methods. A more accurate distinction is that methods exist on a continuum from adaptive to predictive. Agile methods lie on the adaptive side of this continuum. Adaptive methods (Agile Methods), focus on adapting quickly to changing realities. When the needs of a project change, an adaptive team changes as well. An adaptive team will have difficulty describing exactly what will happen in the future. The further away a date is, the more vague an adaptive method will be about what will happen on that date. An adaptive team cannot report exactly what tasks are being done next week, but only which features are planned for next month. When asked about a release six months from now, an adaptive team may only be able to report the mission statement for the release, or a statement of expected value vs. cost.
Predictive methods (Traditional Methods), in contrast, focus on planning the future in detail. A predictive team can report exactly what features and tasks are planned for the entire length of the development process. Predictive teams have difficulty in changing direction. The plan is typically optimized for the original destination and changing direction can require completed work to be started over. Predictive teams will often institute a change control board to ensure that only the most valuable changes are considered.

1. Traditional Methods are called heavy weight methods, while agile methods are called light weight methods.
2. Customer involvement is very less in traditional methods where as in agile methods customer involvement is more.
3. Development with traditional methods comes under the predictive approach, where as development with agile methods comes under the adaptive approach.
4. Traditional approach is suitable for large/small projects, but with the agile methods only the small projects can be done successfully.
5. In traditional approach the documentation is very high, while documentation is least practices activity in agile approach.
6. Traditional approach is a predictive approach which is process oriented but the agile is an adaptive approach that is people oriented.
7. As traditional approach is used for large projects, so the team size is large in this but in agile approach the small projects get handled by small teams.
8. In traditional methods only in the end the organization get the return on investment, where as in the agile approach return on investment comes early in project.

4.4 Conclusion

Software development methodologies have evolved since the 1970s. Agile methodologies came into existence after the need for a light way to do software development in order to accommodate changing requirements environment. Agile methodologies provide some practices that facilitate communication between the developer and the customer, and
under go develop-deliver-feedback cycles, to have more specific view of the requirements, and be ready for any change at any time.

Agile methodologies are not best suited for all projects. When the development team includes mainly beginners or non experienced persons or when difficulty arises in communication between the developer and the customer, agile methodologies will not give the best results. When the development team compromises skilled team members and when there is a strong communication between the developer and the customer, these methodologies exhibit optimum results. When there is a big chance for misunderstanding the exact customer’s requirements, or when the deadlines and budgets are tight, then agile methodologies are among the best software development approaches to apply.

4.5 Comparison between agile methodologies

The main aim of agile methodologies is to deliver what is needed, when it is needed. A set of software development approaches are included in agile methodologies. They share the same basic concepts despite some variations. The main agile methodologies that are being used include XP and SCRUM.

4.5.1 Similarity between XP and Scrum

XP and Scrum have some common features which are as follows:

1. A degree of unpredictability in system specifications is taken into account in both methodologies. In the case of XP rather than to define all system requirements at the outset, a relatively brief planning phase is used. Similarly in SCRUM, planning is performed up to a point, but the understanding that specifications may change during the Sprints allows for rapid adaptation. In SCRUM, this unpredictability is often referred to as chaos.

2. In both methodologies focus is given on producing working software deployments in a short time. The XP practice specifies the need for small working releases, so that as each component of a larger system is created, there is less danger of the overall product failing.
Similarly these functional deliverables are released at the conclusion of each Sprint, and updated however necessary for SCRUM.

3. Stress on frequent communication between team members is given in both the methodologies. For XP, this is crucial when communicating with the on-site customer to clarify system requirements and priorities. When re-factoring the system, communication is fundamental to ensure that all programmers understand the design correctly. Communication is required daily in the form of the scrum meeting in the SCRUM development model. Without frequent communication, errors could go undetected, requirements or priorities misunderstood, or many other such problems. When such rapid development is being performed, proper communication is the only way to prevent outright chaos from overtaking the project.

4.5.2 Differences between XP and Scrum

Despite the frequent overlaps in XP and SCRUM ideologies, there remain several key differences between the two in practice. The fundamental difference between the two can be summarized as follows:

1. SCRUM provides agile management mechanisms whereas the extreme Programming provides engineering practices. No attention is paid to the engineering practices in Scrum, instead focusing on the business of managing the software development. XP, on the other hand, is primarily concerned with those engineering disciplines that will yield the best code. Due to this difference, several distinctions about the two agile methods can be made.

2. Scrum teams typically work in iterations called sprints that are from two weeks to one month long. XP teams typically work in iterations that are one or two weeks long.

3. Changes into sprints are not allowed by Scrum teams. Once the sprint planning meeting is completed and a commitment made to deliver 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 work
hasn’t started by team on a particular feature, a new feature of equivalent size can be swapped into the XP team’s iteration in exchange for the un-started feature.

4. In SCRUM great responsibility is given to the Scrum Master, who manages the development team. He is the central figure who controls the direction of product development. In XP, the practice of collective ownership allows any programmer to modify a section of code when it needs to be fixed. This is accomplished as the need arises, and is not regulated by any central managing figure.

5. In SCRUM, validation of the software is completed at the end of each Sprint during the Sprint review, not at each step within the Sprint while XP requires that the software be validated at all times to the extent that tests are written prior to the actual software.

6. XP also involves a 40-hour week, suggesting that when programmers are more rested, they produce better quality code. The SCRUM process makes no such fine distinction; the workweek could be more or less than 40 hours, depending on the place of business and the employer who specifies the hours worked.

7. Extreme Programming teams work in a strict priority order. Features to be developed are prioritized by the customer 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. However, at some point one of the high priority items may not be a good fit for the sprint being planned—maybe a key person who should work on it will be swamped by work on higher priority items or maybe it makes sense to work on a slightly lower priority item.

8. In XP, the practice of pair programming is one of its most famous features. With two programmers working at the same workstation, the code produced is higher quality. SCRUM development is also silent on this subject, neither advocating nor contesting its use. Such a decision would rest in the hands of the project leader, who would define the engineering methodologies to be used during the Sprint.

9. Agile methods differ in the requirements they satisfy:
   • XP small, co-located teams.
   • SCRUM, not limited to small, co-located teams.
4.5.3 **Advantages of Scrum over XP for Corporations:**

1) It offers certification - corporates can hire people who are “certified” and have their own people certified.

2) Scrum traces its roots back to the Harvard Business Review - so it must be serious.

3) Scrum does not contain the words “extreme” or “programming”. No need to dirty our hands with messy code, keep that stuff as far away as possible

4.5.4 **Advantages of XP over Scrum:**

1) XP actively seeks to address the quality problems from which the most software development teams suffer and which cripple productivity.

2) XP excites the people who actually do the work - dispensing with “resistance to change” in one fell swoop.

After analyzing XP and Scrum carefully it can be concluded that

**XP is a superset of Scrum, and, in my humble opinion, XP is superior.**

4.5.5 **Difference between FDD and XP**

1. XP seem to be better suited for volatile projects, where user requirements are obscure or change often. XP deals with such projects better since it deliberately avoids any activities that are not immediately required for current implementation stage. FDD offers better predictability if requirements for the project are more stable.

2. XP seems to be less scalable than FDD. XP heavily relies on communications with in the team, which becomes harder as team grows.

3. FDD has methods to track project’s progress, which is more appealing in corporate environment.
4. XP does not have the formal design stage. The design may be performed at
the beginning of the iteration and later on as-need basis. FDD has design stage
and design review meeting. During the design stage the developers declare
main classes, methods and properties required to implement the given feature.

5. XP promotes re-factoring. Re-factoring is a process of changing code to make
it better, without breaking the existing functionality. The resulting code has
higher quality. Unlike XP, FDD strongly discourages re-factoring. The main
argument against re-factoring here is that it takes time and does not bring any
value to the customer. The quality of code is addressed during code review
meetings.

6. XP rejects the idea of code ownership. While designated programmers create
the initial implementation, anyone can modify the code. The practice of
browsing/modifying the code of other developers widens knowledge about the
system as a whole. If someone leaves the team, the negative impact of lost
knowledge is minimized. There is a natural restriction on collective code
ownership: FDD encourages strong code ownership. The main idea is that
every developer knows the owned code and better realizes the consequence of
changes. FDD fights the problem of leaving team members from the different
angle.

7. XP does not believe in code documentation. The benefit of FDD is that it
produces the sufficient amount of code documentation.

8. In XP the iteration includes the whole team, while FDD forms the small
volatile teams of 3-5 developers for each iteration

4.5.6 Conclusion regarding XP and FDD

1. To avoid main weakness of waterfall when errors are discovered at later stage
both methodologies successfully utilize iterative development.

2. Both methodologies are lightweight, but XP is lighter: It does not produce code
documentation and keeps code review informal.
3. XP is better suited for the projects with frequently changing or poorly defined user requirements. However, when user requirements are rather stable, it offers higher success repeatability than waterfall.

4. FDD scales better and has a hierarchy within the development team that allows keeping iteration teams small even when the project team is big.

5. FDD offers progress tracking and reporting capabilities. This comforts managers and makes it more attractive for big companies.

6. FDD project might be managed manually, but the project management tools reduce the overhead of keeping the database of features and generating the reports. XP does not seem to require any special tools.

7. FDD is better prepared to work with team where developers’ experience varies. The most experience/productive members become chief programmers.

4.5.7 Comparison between XP and DSDM

1. XP is based on engineering practices where as DSDM is based on engineering as well as management practices.

2. Team Size in XP is 3-16 where as in DSDM the team size is 2-6.

3. XP is more flexible because XP allowed to do changes with in a sprint at any time whereas DSDM is less flexible because it allows to do changes based upon a fixed time line and fixed resources.

4. Length of sprints in XP is 1 or 2 weeks where as length of sprints in DSDM is fixed time.

5. In XP focus is given on usability where as in DSDM more focus is given on fixed time and less on usability.

6. XP is suitable for small projects where as DSDM is suitable for small as well as large projects.
<table>
<thead>
<tr>
<th>Parameters</th>
<th>XP</th>
<th>Scrum</th>
<th>FDD</th>
<th>DSDM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Team Size</td>
<td>3-16</td>
<td>5-9</td>
<td>6-15</td>
<td>2-6</td>
</tr>
<tr>
<td>Practices</td>
<td>Engineering</td>
<td>Management Practices</td>
<td>Feature Based</td>
<td>Based on engineering and management practices</td>
</tr>
<tr>
<td>Flexibility</td>
<td>Allowed to do changes with in a sprint</td>
<td>Don’t allow to do changes with in a sprint.</td>
<td>Allows changes with in a feature</td>
<td>Allowed to change based upon a fixed time line and fixed resources.</td>
</tr>
<tr>
<td>Usability</td>
<td>Focus on usability</td>
<td>Ignores usability</td>
<td>Focus on particular feature</td>
<td>Allowed to change based upon a fixed time line and fixed resources</td>
</tr>
<tr>
<td>Length of sprints</td>
<td>1 or 2 weeks</td>
<td>2 weeks to 1 month</td>
<td>Takes 2 or less weeks .if takes further time, it is decomposed</td>
<td>Fixed time</td>
</tr>
<tr>
<td>Acceptability</td>
<td>High</td>
<td>High</td>
<td>Low</td>
<td>Low</td>
</tr>
<tr>
<td>Validation of software or testing</td>
<td>validation all the times</td>
<td>Validation is completed after a sprint during sprint review not at</td>
<td>At last stage</td>
<td>Testing in all phases of a project</td>
</tr>
</tbody>
</table>
4.5.8 Conclusion

As, in this chapter all the agile methods are explained in detail with their advantages and disadvantages, and after comparing all the agile methodologies finally, we can conclude that the XP is best among all on the basis of various parameters like usability, flexibility, acceptability and suitability. Inspite of being the best among the existing agile methodologies, the XP has some drawbacks which shall be overcome with the proposed agile methodology in the next chapter.