Agility has turned into today's appeal when clarifying a current programming methodology. An agile team can be defined as a brisk team proficient to effectively react to changes. Agile programming advancement alludes to a gathering of software improvement techniques focused around dull work rhythms, where necessities and solutions progress through participation and backing flanked by a self-organizing cross-utilitarian groups. This expression “Agile Method” was invented more recently in February 2001 after the formulation of Agile Manifesto [Beck, (2001)]. Agile methodologies are an alternative to conventional project management viz. Waterfall, or sequential development. In the current software market, it is regularly tricky to anticipate how a product (for instance, a Web-based application) will develop over time. Market circumstances transform quickly, end-client prerequisites advance, and new focused intimidation rise devoid of notice. With a ton of circumstances, experts won't have the capacity to recognize prerequisites completed before the task starts. One must be responsive enough to respond to a condensed business domain. The Agile software development paradigm has been practiced principally due to the indispensable need for more rapid and flexible software. This practice is carried out in a vision to trim down the cost and development time. Agile routines generally push a task administration process that backings incessant assessment and adjustment, an administration rationality that advertises cooperation, responsibility and relationship to oneself. It also includes designing a set of finest practices that permit speedy deliverance of high-quality software, along with a business approach that mastermind in a line improvement with client needs and organizational objectives.
Agile routines accentuate face-to-face correspondence above printed correspondence when the team is collocated. It gives emphasis to working software as the prime gauge of progress. Figure 4.1 describes the iterative processes carried out in an Agile software development. This chapter focuses on the principles and manifestoes of Agile paradigm and provides a concise and basic introduction to the Agile method along with discussion about the existing estimation scenario in this new paradigm.

Figure 4.1: Agile Software Development
4.2 Agile Background

Even though, the original ideas that steer agile development have existed in one form or another for many years, but these plans have packaged into a "development" during last two decades. In crucial, Light or Agile methods were created in a try to overcome evident and concrete shortcomings in conventional programming engineering. The term “Agile” was coined by the 17 proponents and prominent software developers in a meeting, more recently in February 2001 of what was then known as “light methodologies.” The meeting consequence in four levels of conformity among members. [http://agilemanifesto.org].

1. There is a need for routines planned to react to change amid software activities. Further, they agree to the term “Agile” to recognize those techniques. They decided that the expression “light” was not fitting on the grounds that certain tasks would not utilize a "light" methodology yet could even now oblige agility.

2. The second level of conformity was on the four proclamations of the “Agile Manifesto”. These four proclamations catch the center values on which the entire of the Agile structure are built, and the spirit in which they should be put into practice as well.

3. The formulation of 12 Agile Principles was the next conformity level. Within these announcements the qualities are fleshed out in included detail and given more tangible meaning.

4. The final level of conformity was that, agreement at an added detailed level was outside their grasp at the time. They were contented to leave that last level for freedom to describe the known methods of Agile in its own way.

In view of the fact that software specialists are not machines. They exhibit incredible variety in meeting expectations, styles; inventiveness, consistency tidiness, the Agile methods have become an admired topic in software development communities, giving rise to much mystification and argument.
4.3 Agile Manifestoes

The Agile Manifesto was printed in 2001, and offers a good quality, decisive point for examining the Agile methods and understanding the foundation on which they were built [Beck, (2001)].

Figure 4.2: Agile Manifestoes for software development

Figure 4.2 highlights the manifestoes of agile paradigm published in 2001. The entire Agile based development rotates around these manifestos. The four statements stated above of the Agile Manifesto are uncomplicated and straightforward. The first and third are regarding communication. They spotlight the fact that software development takes
place in a people centric environment. Though people are not predictable components, communication amongst those people is a key success factor. The second and fourth statements are about the sensible view of what adds up to success in a software project; it has to produce software that congregates the (varying) needs of the ultimate users. The four Agile Manifestoes are quite high level and intangible. They have a lot of room for elucidation and do not give much in the way of guidance.

4.4 Agile Principles

The Agile Manifesto discussed in the previous section is fairly high level and abstractions. The Agile Alliance characterizes 12 agility principles which flesh out the ideas articulated in the Manifesto. These 12 principles were documented during the February 2001 summit. They present a much more concrete view of the type of activities that comprise the various Agile software development method, Figure 4.3 illustrates how the 12 Principles of Agile methods embrace a number of ideas that are radically different from those embodied by more established methods.

4.4.1 Principle - 1

The unique peculiarity of first principle is the way by which strategies seek the objective of customer satisfaction. Agile methods tend to shun long forthright requirements design and development actions. Modestly, in the wake of setting a fitting establishment, they chose for creating operational programming as quickly as would be prudent. In real meaning, Agile methods substitute the conventional necessities and outline periods of advancement through ahead of schedule verification of-idea exercises. These methods appear really likened to prototyping projects, however with the distinction that the consequential software product is delivered for make use of, as opposed to being discarded.
Figure 4.3: Principles of Agile methods [http://agilemanifesto.org/principles.html]
4.4.2 Principle - 2

This is one of the founding principles on which the Agile scenario is constructed and the intention why the name “Agile” was adopted. This rule concedes the reality that change is unsurprising and creates the reasoning that advertising customary change is an in addition to of the Agile techniques for their clientele. Instead of attempting to control change, the Agile Methods are planned to empower change completely through the project life cycle.

4.4.3 Principle - 3

It is unambiguous that the augmentations on Agile ventures should be as little as could be expected under the circumstances and goes so far as to say that a week or two is not exorbitantly short for an addition. It likewise lays a farthest point of confinement that may be unanticipated to a lot of people, in light of the fact that advancement periods of short of what three months are normally not seen.

4.4.4 Principle - 4

The agile methods entail cooperation and collaboration between the product advancement group and the included stakeholders in the undertaking. A most central part in Agile activities are given to the last user of the framework being created. However, all different stakeholders have huge parts and are required to intermingle with the development group all the time.

4.4.5 Principle - 5

The Agile methods are all lies on the hypothesis that software advancement colleagues are all roused, capable, proficient, propped up by the association, and engage to perform their jobs. In the meantime, every technique raises an environment that is intrinsically spurring to the specialized staff and has the normal impact of building each one colleague's fitness to toward oneself spur and self administer.
4.4.6 Principle - 6

This principle elucidates the tendency of Agile methods to have a little written communication. All recommends vis-à-vis correspondence as the key method for imparting data, regularly append aging it with tools for example whiteboards. The correspondence is often sorted out as “information radiators,” which are outwardly accessible posting of information to which members of the development team can refer each time it is needed.

4.4.7 Principle - 7

The principle enlightens diverse path in which the Agile techniques pound against documentation. The majority of software projects will make out the acknowledgement of a requirements or design specification, as a point of interest indicating important advancement. Various Agile methods refuse that thought. They have a tendency to put slight significance on the project’s interim products and focus altogether on the ensuing product. Certainly, as operational software is delivered repeatedly on an Agile project. It becomes a sensible measure of progress.

4.4.8 Principle - 8

The Agile methods insurgent against employee overtime and their norms and argue in favor of “sustainable” schedule. Through this principle they don't recommend to anticipate 30 hours for every week, then 60 hours; relatively they mean to revise the expectation of what can be consummated so as to minimize the overtime. But according to this principle, the way of the Agile methods has an inclination to level out the requests on the advancement group so that a more practical pace is regularly kept up.

4.4.9 Principle - 9

This principle is possibly a standout amongst the most critical of the Agile methods. All Agile methods stressed upon making sure that the nature of the specialized work is maintained in an abnormal state. The variety of Agile strategies attain this in diverse
ways, yet everybody has a solid quality part. This principle offers an elevating counterpoint to the normal discernment that the designer’s employment is to create programs and another person is responsible for its right meeting expectations. In the Agile strategies, advancement groups catch the essential obligation to make certain that what is delivered is right and satisfactory.

4.4.10 Principle - 10

Prior to the existence of Agile Manifestos, the methods that go under the "Agile" Umbrella were for the most part alluded to as "light" strategies. The expression "light" was dropped out of recognizing that certain application spaces involve heavier forms. Indeed thus, the Agile routines at all times anxiety utilizing the lightest possible procedure.

4.4.11 Principle - 11

This principle stress upon the Self management of Agile teams. It is a trait of Agile methods and a repercussion against the charge and-control systems used to handle different programming tasks. There are emergent evidences that supervised toward oneself groups are really truly proficient. However, moving to regulated toward oneself groups involves considerable changes all through the association. The key change happens in the positions of the administration, where manager’s parts must grow in the direction of training heading, as a specialist job develop to incorporate self management. These adjustments are not easy to impact and oblige that numerous individuals find new practices and abilities

4.4.12 Principle - 12

The concept of manifestation is well-known in the software industry. The Agile methods take on this imperative action as their key method for enhancing their methods. A couple of Agile strategies take this, to a great extent by doing manifestation at the end of every iteration of product development, consequent in lessons learned and flawlessness in methodology persistently.
4.5 The Human Factor

The extremely significant value in the Agile manifesto sketch s a line in the sand. Promoters of agile software development clearly state their confidence that Human/People are of greater importance informative software project success instead of processes or tools. According to a researcher, “Agile development focuses on the talents and skills of individuals, molding the process to specific people and teams” [Cockburn and High smith, (2001)]. The critical tip in this announcement is that the methodology molds to the necessities of the individuals and the group, not the other path around. On the off chance that software development team allies are to coerce the attributes of the progression which is helpful to assemble programming, a set of different imperative qualities must present amidst the individuals on an agile group and the group itself.

The Agilest are not only advocates, the people. Human or people have been within the software industry for decades. The software engineering community believes that if one compose the team of the right people, and attends to their needs, then individuals will be capable to succeed no matter what. Some say that disciplined processes and structured tools can also get in the way of project success. Figure 4.4 illustrates the three interconnected aspect i.e. People, Process and tools for project’s success. To make one aspect week or underestimated than the other two would make the project unstable and eliminating any of these would cause the project to fail.

Figure 4.4: Role of People, Process and Tools in Agile Environment [Cockburn, (2007)]
People are considered to be most valued resource. Industries that create intellectual property, people have a superlative role, as observed in figure 4.4. Process cannot imagine and create. Tools cannot apply intelligence. Converting ideas into running software requires people. People envision, people understand, people visualize, people imagine what they suppose to build, and then people transform that vision into actuality. Writing sophisticated and quality software cannot be imagined without people.

However, figure 4.5 also highlights the deficiencies of human or people. People can make errors that result in faulty software product. People fail to remember things so that their solutions are imperfect. People may visualize the future inaccurately to that their plans and estimates may be poor. At one time People can maintain only a restricted amount of information in their brain, so they may miss the outcomes of their decisions. People misapprehend what other states so that information is vanished in communication. People may remember more or less so that the facts are subject to disagreement. Human resources are expensive and may work unhurriedly resulting to over budgeting and over scheduling of projects. People are tired and bored of something by repetition so that some tasks are overlooked or performed badly.
For the reason mentioned above, humans by themselves are not enough to guarantee a successful software project. Even a best software development team will have an unsure chance of success. All human resources require the support that is provided by processes and tools to alleviate for their weaknesses so they can execute their best work.

### 4.6 Agile Team Size

The entire Agile methods appear best fitted to the small size of software project teams, i.e., Teams size lower than 15 persons. Even though some methods do not unequivocally place a bound on team size, their inclinations for face to face communication over textual communication places a practical limit on team size, figure 4.6 illustrates the applicability of agile methods with respect to the team size. According to Agile principles merely a restricted number of people can energetically take part in a meeting and despite of whether an Agile method call for regular team meetings or not, every Agile method involve constant communication with all team members.

![Applicability of Agile Method](image)

Figure 4.6: Agile Team Size
Increasing Scope of the software project entails team size to be bigger. The Agile methods are not inevitably inapt for bigger team size. In projects of 50 human resources or more, individuals are almost always clustered into smaller sub teams for convenience and practical purposes. Each sub team will repeatedly operate somewhat separately from the others and join together its work products at appropriate mark. Such an environment requires the use of agile method by one or more of the sub team for the execution of the work.

**4.7 Agile Methods**

Software Development Methods are present in a gamut from adaptive to predictive [Boehm and Turner, (2004)]. Though, there exist more than 12 Agile methods, but famous and well accepted agile software development methods are depicted in figure 4.7 and described briefly in the subsequent sections.
4.7.1 Adaptive Software Development (ASD)

This Agile method has been proposed by Highsmith. It is based on a collaborative learning cycle. He analyzes a software project team as a complex, versatile framework that comprises of agents (team members as well as other stakeholders), environments and emergent outcomes (developed product). An ASD “life cycle” is described through three phase’s i.e. Speculate, collaborate, and learn as shown in figure 4.8 [Highsmith, (1999)].

![Diagram of Collaborate, Speculate, Learn cycle]

Figure 4.8: Adaptive S/W Development

4.7.1.1 Speculate

The project’s initial step in speculation is an instigation workshop and adaptive cycle planning is conducted. During instigation, the whole project team and the stakeholders or customer establish the project’s initiation information, including the objectives and project constraints, customer’s mission, the system requirements, the organization for the task, preliminary estimates of product size and scope and key risks to the project.
The initial cycle of the project begins with the speculative practice usually called as “planning”. This merely incorporates following step:

1. Find out the project time box.
2. Decide the best possible number of iterations and the time-box intended for every iteration.
3. Writing of a statement of purpose or aim for every cycle.
4. Allocate principal components to cycles.
5. Allocate technology and support mechanism to iterations.
6. Build up a list of project task.

In each succeeding project cycle, Adaptive Cycle Planning consists of revisiting and adjusting these things as needed, based on growth to date and what has been learned in previous cycles.

4.7.1.2 Collaborate

This is the phase where the real work carried out. The substance of this phase of every cycle is planned in the Adaptive Cycle Planning phase and completed to the extent possible within the intended time box.

Collaboration is applied among motivated people in a manner that proliferate their ability and innovative conclusion past their total numbers. This methodology is an unending topic in dominant part of Agile techniques. However, collaboration is definitely not a straightforward task, it includes communication and teamwork, and emphasizes individualism as well. Individual creativeness plays a significant role in collaborative thinking.

4.7.1.3 Learn

The end of each cycle is marked by learning actions, where the project team get close into progress to date and gathers the information they need to do any rescheduling at the inception of the next cycle.
According to Highsmith, software developers habitually overrate their own understanding with the project. In order to enhance the level of genuine comprehension of individual in the team, “Learning” is essential and practiced. Learning in ASD is accomplished through below mentioned three ways

1. **Focus groups** - since the ultimate objective of an ASD project is to congregate on a system that meets up its business intention, evaluating the results of each cycle of the last users is a vital learning activity. A joint workshop is conducted and during this workshop, developers would express and elucidate what has been accomplished so far. Then end user would undertake using the software and give the development team their responses to it.

2. **Technical reviews** – The primary objective of these reviews is to identify defects in the work products of the succession, but they have a secondary advantage of ensuring that each team member is familiar with all the code that has been written.

3. **Postmortems** - The concluding step in each cycle is the postmortem, where team members assess the efficacy of the process they have used in addition to the project’s performance against its plan. The outcomes of the postmortem are supplied back to the planning phase of the subsequent cycle.

The ASD belief has advantage in spite of of the development model applied. On the whole, ASD burdens on the motion of sorting toward oneself out groups and interpersonal collaboration. Individual and group learning set forward software project groups with an elevated plausibility of achievement.

**4.7.2 Dynamic system development method (DSDM)**

DSDM is one of the important software project development methods under the Agile umbrella that offers a delivery skeleton, intended for organizing and keeping up programming systems that requires unbending schedule demands from start to finish.
through incremental prototyping in an illicit project turf. DSDM initially discharged in 1994, and needed to invest with some control in the fast or rapid application development method. DSDM end up being a bland strategy on the way to project management along with result delivery. The thought of DSDM is obtained from a customized adaptation of the Pareto rule i.e. Eighty percent of an application might be conveyed in twenty percent of the time it would catch to convey the entire application. DSDM is referred as an incremental and iterative programming process that holds standards of Agile development and provides an ordered set of activities with feed–forward and feedback loops, but it permits enough freedom for any particular project to delineate exactly how individual activities bring together to define the project’s life cycle. [http://www.dsdm.org/]

“The DSDM life cycle” model is described by the DSDM Consortium is shown in figure 4.9. It is also called DSDM process model that typically includes steps for assessing feasibility, studying the business needs, and functional modeling. Programming is covered in the “Design and Build” step.

1. **Feasibility and business study** set up the fundamental business chuck and restriction concerning the software to be created and after that survey, whether the undertaking is an attainable hopeful for the DSDM process. Business study finds out the practical and data prerequisites that will allow the application to make accessible business esteem. This additionally, characterizes the essential application structural outline and recognizes the practical requirements for the application.

2. **Functional model iteration or exploration** generates a series of incremental models that expresses usefulness intended for the client. The reason for using such iterative cycle is to collect extra prerequisites by getting reactions from clients who practice the model.
3. **Design and engineering iteration** reevaluates models created amid investigation to make sure the building of the model which empowers it to supply prepared business esteem for clients.

4. **Incremental Implementation** deploys the most recent software increment into the operational environment. The precondition for this phase includes that; the progressions may perhaps be demanded as the augmentation is established. In any
cases, DSDM happens by coming back to the useful model cycle or investigation action.

4.7.3 Extreme Programming (XP)

XP stands for extreme programming and was publicized by Kent Beck. This is the most commonly known Agile method. It focuses on the development rather than managerial parts of software projects. XP can be defined as a methodology for software development that picks up the responsiveness and quality of the software to varying customer requirements. Being an agile software development method, it supports and promotes repeated "release" in smallest reasonable chunks, which is intended to provide demonstrable value, enhance output and present checkpoints where new client prerequisites might be received [Abrahamsson, (2003)].

An XP project begins with a release planning phase, pursued by a number of iterations, each of which terminates with user acceptance testing. When the application software has adequate features to satisfy customers, the XP team terminates iteration and finally releases the software [http://www.extremeprogramming.org/].

XP utilizes an object-oriented approach, as its perfect advancement situation incorporates a set of standards and practices that emerges inside the connection of six key development exercises: planning, analysis, design, implementation, testing and maintenance. Figure 4.10 demonstrates the XP process and Key XP occasions are delineated in the accompanying segment.
4.7.3.1 Planning

XP projects initiate with a planning phase, trailed by several iterations. The activity in this phase starts with “listening” i.e. A necessities gathering movement that permits the specialized persons of the XP group to distinguish the business viewpoint for the software product. Listening results in the creation of “user stories” that speak to oblige gimmicks and usefulness for the application to be designed and developed. With the progress of the work the customer or client can append stories, improve the value of a previously submitted story, isolate stories, or kill them. The reassessment of all enduring releases and changing of the plans in like manner is accomplished by the XP development team.
4.7.3.2 Analysis

XP analysis works on the outcome of the planning phase, this iterative phase results in the specification of software’s functional characteristics, specify the interface of software with further elements of the system, and launch constriction that software should follow. An analysis phase permits XP team to expand on fundamental necessities secured in the origin, negotiation and elicitation errands.

4.7.3.3 Design

This practice is the heart of XP’s value of “simplicity” where a straightforward outline is constantly supported above more convoluted representation. The design phase also endows with implementation guidance for a story precisely. In case of any complex story design XP suggests the immediate formation of a functional model of concerned part of the configuration; the outline model is executed and assessed. XP stands upon philosophy that this simple design practice will cause less revision than designing for the future.

4.7.3.4 Implementation

Following the story development and preparatory configuration work the XP group does not specifically move to execution or coding, instead they create an arrangement of unit tests that will practice each of the stories that is to be incorporated in the product increase. Coding the XP methodology comprises of code a bit, test a bit, code somewhat, and test a bit. Upon creation of the unit test, the developer focuses on what ought to be executed to clear the test. At the point when the code is finished, it may be unit-tested without delay, along these lines giving a quick reaction to the software engineers. The most visible practice and key notion during this phase is pair programming. XP advocates that pair of programmer working collectively at one machine unit to produce code for a story. The pair has formed and change dynamically all over the project, according to the requirements of each story. This offer a tool for constant problem solving and continuous quality guarantee (the code is assessed as it is generated). It additionally keeps the
engineers concentrated on the issue close by. In practice, every individual undertakes a marginally distinctive part.

**4.7.3.5 Testing**

One of the XP’s essential values is “Test First”. According to this value, prior to a pair of programmers write a single line of code, they have to implement the automated test that will be needed to verify the story they are about to write. This promotes a regression testing methodology each one time the code is changed. Since the individual unit tests are orchestrated into a "widespread testing suite", integration and acceptance testing of the framework can occur step by step. This offers the XP group with a continuous evidence of advancement furthermore, can caution early if wrong things.

**4.7.4 Feature Driven Development (FDD)**

FDD was introduced in 1997 as a valuable software, development model in favor of object-oriented programming, designing by Peter Coad and his team. FDD diverges from other Agile methods in its focus on straight design and planning. From the FDD perspective, a “feature” is a client-valued function that might be executed within a short span i.e. Two weeks [Coad et al., (1999)].

As shown in figure 4.11, the overall model, feature list and planning are finished once at the commencement of the project, and iterations are effectively an incremental building of identified features.
Similar to other agile approaches, FDD accepts a values that

1. Accentuate collaboration amongst individuals of a FDD group.
2. Deals with issues and multifaceted nature of the project by means of feature based disintegration pursued by the assimilation of software augmentation in terms of increments.
3. Communication of the technological aspect utilizing oral, graphics, and content based means.
4. Makes intense use of Inspections to make sure the quality of the designs and code that are built.
5. Regular builds are done to uphold a current system encompass of all the features that have been developed to date.
4.7.4.1 Reporting of results

FDD exercises a distinctive mechanism intended for tracking and reporting the project’s status. This mechanism utilizes the project’s feature list and feature development milestones along with a number of weighting factors for those milestones. Equation 4.1 depicts the formula for obtaining the current status of the project.

\[
\text{Feature Value} = \sum \text{(weights of completed milestones)}
\]

\[
\text{Project Status} = \frac{\sum \text{Feature Value}}{5} \quad \left[ \begin{array}{c} 0.0 \leq \text{Feature not yet worked on} \leq 0.0 \\ 1.0 \leq \text{Feature that is complete} \leq 1.0 \\ 0 \leq \text{Feature in progress} \leq 1.0 \end{array} \right]
\]

4.7.5 Lean Software Development (LSD)

LSD is not a hard core software development method. Rather, it can be defined as a collection of values along with tools, that a software development company can utilize in order to make more Lean software.

LSD has tailored the software engineering standards of lean assembling to the community of software developers [Cockburn A., (2007)]. Lean Development is typified by 7 lean principles that are convoluted into 22 Lean Software Development tools as shown in figure 4.12.
Each one of the illustrated principles and corresponding tools could be adjusted to the production process. Case in point, wipe out waste inside the setting of an agile project can be taken to mean, looking for waste in these parts of any development process.

Partly done work (unimplemented designs)

Spare processes (Steps in the process that do not add value)

Extra Features (stuffs for which the customer did not ask)
Task switch (people assigned to multiple projects)

Defects

4.7.6 Scrum

The scrum software development method came into existence in the early 1990s as a light footed technique. It was introduced by Jeff Sutherland as a way for organizing software development that can be enclosed about some explicit technology, including software. The term “Scrum” is taken from the game of Rugby and refers to a tactic used to get a ball back into play [www.scrum.org].

As with other Agile methods, Scrum standards are unswerving with the agile declaration and are utilized to guide improvement exercises inside a process that absorb the accompanying framework. The general stream of the Scrum methodology is indicated in Figure 4.13. Scrum set up a term “sprint” which is a preset time duration for which the Agile team works [Schwaber and Beedle, (2002)]. The Scrum development method gives attention to on managing sprints. Prior to each sprint begins; the team prepares the sprint, spotting the backlog items and assigning teams to these items. Teams develop, wrap up, review, and amend each of the backlog items.

Product backlog is the entire list of requirements, enhancement requests, including bugs, and usability and performance upgrading that are not presently in the product release. Throughout development, the team finds out the changes necessary to implement a backlog item. This is followed by writing the code, testing it, and documenting the changes. During wrap, the team builds the executable necessary to show the changes. Within review, the team displays the new features, includes new backlog items, and evaluates risk. At last, the team combines data from the review to update the changes as required.
4.7.7 Crystal Clear

It is a constituent of the Crystal group of strategies as clarified via Alistair Cockburn in 1998 and is viewed as a case of an agile methodology. Cockburn thinks that there is a
nonexistent of any universally accepted measure that fits in all types of development process. Therefore, the diverse methods are allotted colors prearranged in rising cloudiness; the main variant of Agile is Crystal Clear, pursued by Crystal Yellow, Crystal Orange, furthermore Crystal Red [Cockburn, (2005)].

Crystal can be characterized as a product development approach that burdens on "mobility" through Cockburn portrays as "an asset restricted, common round of creation and correspondence what Cockburn characterizes as “a resource limited, mutual game of creation and communication, with a prime target of conveying important, operational software. The secondary objective is the preparation for the subsequent game”. In fact the Crystal family is a collection of model agile methods that have been confirmed adequate for different mixed bag of software projects. The Agile team should pick the best suitable Crystal approach which is most fitting for successful and timely completion of their project. Crystal Clear is a streamlining of Crystal that may be connected when the development group comprises of 3 to 8 individuals sitting in the similar room or abutting working environments. The Crystal family of methodologies concentrates on the effectiveness and habitability as factors of project safety [Boehm and Turner, (2004)]. Crystal strategies put significant stress on correspondence among individuals concerned to the venture. Common qualities of Crystal methodologies are as follows.

- Observance to the standards of agile improvement
- Iterative incremental process, with every increment should not exceed four months.
- Refuse to support dispersed teams.
- Collocation of the peoples involved in the project.
- Easy access to expert users
- Reliance on effective information flow among team-members for successful execution.
- Personal safety
Automatic tests, frequent integration and configuration management.

4.7.8 Test-driven development (TDD)

TDD is an evolutionary programming advancement, practice under the umbrella of Agile standard. It is an advanced strategy of utilizing automated test cases to induce software design and force decoupling of dependencies. It emphasizes on a great degree of small development cycle: in the first step the Agile expert is supposed to write a preset test case that describes a requisite development or new function, subsequently creates the enough amount of code to pass that test, and at last refactoring is applied to the new code to acceptable standards [Beck, (2003)]. TDD is being hastily accepted by agile software professionals in the application source code development and is yet being received by Agile DBAs for database development. TDD should be seen as corresponding to Agile Model Driven Development (AMDD) technique and the two could be misused together. A cons of TDD is that the ensuing tests are operational cases for citing the code, therefore giving a working example of the code. TDD works amazingly well in practice and it is something that all product engineers began embracing it.

4.7.9 Agile Unified Process (AUP)

AUP is proposed and developed by Scott Ambler is a streamlined adjustment of the Rational Unified Process (RUP) [Waters, (2008)]. It exemplifies a clear and straightforward methodology to create programming projects making use of agile techniques and concepts. The AUP applies agile techniques together with, Agile Modeling, database refactoring agile and change management, to get better output. As an important Agile method AUP consents to a "serial in the expansive" and "iterative in the little" thought for building programming frameworks by implementing the standard Unified process activities i.e. Setting up, elaboration, building, and transition. AUP presents a linear progression of software engineering activities (figure 4.14) that allows an AUP group to envisage the entire flow of process in support of a software project. Inside each of the actions, the agile team members are obliged to make iterations so as to
accomplish deftness and to convey utilizable programming augmentations to the customers as quickly as would be prudent [Ambler, (2006)].

**Figure 4.14 : AUP Iteration Activities**

### 4.8 Agile Cost Estimation Scenario

Size and Effort estimation, is a key activity of any type of software project management and it gets to be exceptionally perplexing in the event of unpredictable requirements. Agile methods are much unbeaten in unstable requirements because of their enhanced practices. Agile prescribes collective improvement, meeting evolving requirements with working software and developing application naturally from various emphases [Cockburn, (2007)]. Because of vulnerabilities in the requirement and special web development characteristic, these strategies don't have an enormous forthright in the cost estimation of complete software with least adjustment. Agile Estimation Methods are usually stood upon analogy and expert opinion. In spite of the fact that numerous groups
in practice do agile estimation as vital part of their task, it is exceptionally regular to see the basics driving the estimation are disregarded and move towards giving as exact an assessment as could be allowed. The idea of "Relative" estimate converts to "Outright". Conventional ways of estimation following FP, COCOMO or other methods was all pointing on providing estimates which were "Absolute" in nature. Conversely, agile philosophy encourages a "Relative" nature of estimates aligning with the very depiction of Estimation.

### 4.8.1 Agile Estimation: A Decomposition Approach

Since the prerequisites (requirements) for an agile project are characterized by a collection of client scenarios (for instance, user "stories" in Extreme Programming), it is conceivable to create an estimation approach that is casual, sensibly trained, and compelling inside the context of project planning for every product increment. Agile estimation utilizes a decomposition approach [Pressman, (2010)], that includes the accompanying steps:

1. Every client scenario is viewed as independently for estimation purposes.

2. The scenario is disintegrated into the set of software engineering undertakings (tasks) that will be obliged to create it.

3(a). The effort needed for each one undertaking is evaluated independently. Note: Estimation can be focused around authentic information, an algorithmic model, or "experience."

3(b). On the other hand, the "volume" of the scenario can be assessed in LOC, FP, or some other volume-oriented metrics.
4(a). Estimates for each one undertaking are summed to make an overall estimate for the scenario.

4(b). On the other hand, the volume estimate for the scenario is deciphered into effort utilizing historical information.

5. The effort estimates for all scenario that are to be executed for a given programming increment are summed to create the effort estimate for the increment.

4.8.1 Agile Principles for Estimation

Since the agile end up being additionally intriguing to be embraced as most recent process in the programming development procedure, it is basic to take a gander at how estimation is made. The agile strategy offers uncomplicated procedure contrast with traditional systems. Through agile, the product development gets to be clearer and less demanding [Rees, (2008)], [Stephen, (2009)], [Chandra et al., (2009)]. Standard Principles for Agile Estimation are as follows [Steindl and Krogdahl, (2005)].

I. If estimating is multifaceted, the Agile methodology upgrades the input.
   Condense the estimating time to input about rightness of estimate.
   Raise the recurrence of estimating.
   Sketch out alternatives and get input from the client before doing nitty-gritty estimation

II. In the event that the estimates and necessities are not by any stretch of the imagination reasonable, the Agile estimation group creates
   Diverse and complex appraisals
   Communicate the stipulations of the estimates instead of simply the numbers.
Discussion on the suspicions with the client or stakeholder may be carried out so that, the client can give input to better bring into line the Agile team's liberal with the business drivers.

III. Authenticate estimates by matching up to them with other existing indistinguishable assessments/ experience.

4.9 Chapter Summery

In a present economy Software engineering and software project management are vibrant and rapidly changing occupation. New methodologies have been sprouting on an average of about two per year for the last decade. Owing to the well-known failure of software projects, many practitioners have been seeking unconventional approaches that permits them to go on with agile to delineate, versatile, lean methods that can address the prerequisites of current business. The origin of the set of “Agile” methods can be marked out to the publication of the “Agile manifesto”. This manifesto emphasizes on four key issues i.e. The significance of self-organizing development groups that have control over the task they carry out, communication as well as cooperation between team affiliates and in the middle of experts and their clients, an acknowledgment that change corresponds to an opportunity, and a stress on quick delivery of product that convince the customer. These key issues are adopted by different Agile methods in order to provide running versions of software and shaping the ultimate naturally from various cycles to the prospective clients.

The core spirit of Agile development scenario is that customer requirements for software projects will all the time be fluid and varying. In view of the fact that the Agile development has augmented to the point where perchance 15 percent of software applications below 1000 function points in size now make use of some of the Agile concepts [Jones, (2007)]. The entire range of agile process models discussed in this chapter addresses the significance of people’s factor through Agile Principles of
motivated individuals and self managing teams in effectively similar ways. For instance, an iterative process is followed by ASD which adds adaptive cycle planning, moderately scrupulous requirement gathering techniques, along with an iterative improvement cycle that consists of client focus and official technical assessment as a concurrent feedback system.

XP recommends a various powerful and new procedures that permit Agile experts to make recurrent releases of the software so as to deliver functionalities and characteristics that have been explained and afterward prioritized by stakeholders. Scrum emphasizes the utilization of programming methodology designs that have built solid for programming activities with hardened time plans, business criticality and changing requirements. DSDM helps the utilization of time-box conception and advice that just sufficient work is essential for every software product augmentation to make conceivable development with the resulting addition. FDD is fairly more "formal" among other nimble strategies. In spite of the fact that it maintains nimbleness by centering the Agile project team on the building of peculiarities— a customer esteemed job that could be executed from within two weeks. LSD has adjusted the standards of lean assembling to the universe of software engineering. AUP undertakes a "serial in the substantial" and "iterative in the little" thought for the quick development of software.

The Agile approach is well-liked and admired by its practitioners and has normally been flourishing when deployed for appropriate projects. Even though Agile methods commenced with small projects, efforts are being made to extend the approach to bigger application. As per the principles and nature of Agile approach its applicability is most suited to web based projects. Web based applications are relatively small in size and perhaps the most interesting and swiftly changing field in the era of software engineering. The technology of building web applications and the intricacy of web applications, both are shifting day by day. The statistics of web based applications, increasing exponentially. There is a rapid advancement in the methods of designing web based application. Web based design and development methodologies run the extent from traditional waterfall approaches through the leanest of Agile approaches. For the reason of the unpredictability of web-based applications and their development approaches, it is
something difficult to derive thumb rules that can be applied universally. Moreover, the
greater part of the web application has not devoted enough effort or time for collecting past
data or performing process reviews. There is an inevitable lag between the first
deployment of the new method, and the capability to predict that method’s impact on
efficiency, costs, schedules, and maintenance.