Chapter 1 - Introduction

This chapter presents an overview of traditional software development method, such as Waterfall Model and compares it with Agile Software Development approach. This is followed by the description of Agile Manifesto, twelve Agile Principles and explanation on the background, philosophy, scope, features and benefits; roles and responsibilities; and limitation of various Agile software development methodologies in detail.

1.1 Agile Software Development Vs. Traditional Software Development

The Waterfall Model is a traditional engineering approach applied to software engineering development projects. It is a sequential design practice, in which process flows downwards (like a waterfall) through the phases of conception, planning, requirement analysis, design, testing, implementation and maintenance. In waterfall model, the main emphasis is laid on planning, schedules, budgets and deployment of an entire system at one time. For this purpose, a tight control is maintained over the life of software development project through extensive documentation and formal reviews followed by approval/signoff by the customer that occurs at the end of each phase and before beginning the next phase.

The waterfall methodology normally doesn’t allow complete moving on the next stage unless the previous stage is fully completed and verified. However there are few advantages and pitfalls in this methodology. Due to Big Design Up Front approach of waterfall model, large projects runs over schedule and over budget that fails to deliver on requirements which is a major drawback of this model.

Agile software development approach offers solution to the drawbacks of the waterfall model. Instead of using sequential design or process intensive approach, an Agile software development method follows iterative and incremental approach in a highly collaborative manner to build high-quality software in an effective budget and schedule control allowing projects to adopt the changes in user requirement rapidly. The Agile methodologies focus on delivering the smallest working piece of functionality as early as possible and constantly recuperating by accumulating additional functionalities during the project lifecycle.
The Agile approach helps in minimizing and mitigating the overall risk, and lets the project to adopt the changes fast and does not necessitate a requirements freeze upfront unlike waterfall model. The project is carried out in iterations, typically one to four weeks. The industry mainly chooses Agile software development process as it is highly result oriented. Agile methodologies accentuate effectual communication over well written documents and big design up front. Depending upon business priority, these features are assigned to releases, which are tied to iterations. Agile techniques give emphasis to working software as the essential measure of progress. The key characteristics of the Agile methodology are delivering frequently, incremental and iterative approach, less defects, continuous testing and integration, collaborative approach and maximum return on investment (ROI).

**AGILE MANIFESTO**

In recent years, with the growing competence of software market, researchers are in search of more flexible practices that can be used to adjust to dynamic situations where customer requirements are changing over time. In 2001, the Agile manifesto established the approach now known as Agile software development process by 17 software developers (Beck, et al., 2001). The Agile Manifesto is stated as:

"Individuals and their interactions, over processes and tools
Delivering working software, over comprehensive documentation
Customer collaboration, over contract negotiation
Responding to change, over following a plan"

Separated by commas, the phrases on the right side of the comma relate to value, and, the terms on the left describe the Agile philosophy. Studying each of these values aids in acquisition of knowledge of the Agile process philosophy while divulging how application of the philosophy to defined methodologies will improve software development map it with current unpredictable markets.

**AGILE PRINCIPLES**
The Agile Alliance also documented the principles that underlie the manifesto. Agile methods are principle-based, rather than rule-based and have predefined rules regarding the roles, relationships, and activities. The principles that guide the software developers comprising the team and project manager include:

1) Customer satisfaction through early and continuous delivery of valuable software is the highest priority.
2) Agile processes harness change for the customer's competitive advantage and hence are open to changing requirements, even late in the development process.
3) Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4) Necessarily involve business people and developers so as to work together on a daily basis all through the project.
5) Build projects with motivated individuals. Provide them the necessary setting and support they need, and have confidence in them to get the task done.
6) Convey information in the most proficient and effective method to and within a development team preferably through face-to-face conversation.
7) Working software is the primary measure of progress.
8) Agile processes encourage sustainable development. The sponsors, developers, and users should be able to keep up an unvarying tempo forever.
9) Continuous attention to technical quality and superior design enhances agility.
10) Simplicity, the art of maximizing the amount of work not done is essential.
11) To achieve best architectures, requirements, and designs from self-organizing teams.
12) On a regular basis, preferably at fixed intervals, the team reflects on how to develop into a more effective team then regulates and adjusts its activities consequently.

This section aims to systematically review the existing literature on Agile software development techniques and to understand the features, benefits and challenges of using various Agile techniques.

AGILE METHODOLOGIES
Several Agile techniques have been proposed and used by researchers in difference domains. These Agile methodologies share common principles among themselves but differ in practices. This section identifies some of the well-known existing Agile software development methods and their objectives. These are described in detail below:

1) Extreme Programming (XP)
2) Scrum
3) Lean Software Development (LSD)
4) Kanban
5) Adaptive Software Development (ASD)
6) Feature Driven Development (FDD)
7) Dynamic System Development Method (DSDM)
8) Agile Modeling (AM)
9) Crystal
10) Agile Unified Process (AUP)

**EXTREME PROGRAMMING**

**Background**
It was introduced in 1998 by Kent Beck, Ron Jeffries, Ward Cunnigham based on experience from C3 project (Beck *et al.*, 2000).

**Philosophy**
Extreme Programming (XP) is a well-known and a light weight discipline of software development that focuses on engineering practices. XP seeks to enable successful software development regardless of ambiguous or continuously changing software requirements. It is a system of practices which is intended to improve software quality and quickly addresses the changing customer requirements to meet business needs. It comprises collection of informal requirements from on-site clients, arranges teams of pair programmers, developing simple designs, continuous refactoring, and continuous integration and testing; and advocates frequent releases in short development cycles that
improves productivity as well as introduces checkpoints where new customer requirements can be embraced.

**Scope**

XP is best suited for projects that require collocated teams of small to medium size team. On the project side XP is meant for assignments where the requirements are unstable and unpredictable.

**Features and Benefits**

**Team Size:** 2 to 12 members and preferably co-located  
**Iteration Length:** Usually 1 to 3 weeks  
**XP Phases:** It has 6 phases: Exploration Phase (write story for current iteration), Iteration Planning Phase (Prioritize Stories, effort and resource estimates), Iteration to release Phase (Analysis, design, coding, testing), Production Phase (Rigors testing), Maintenance Phase (Customer supports, release for customer use), Death Phase (No more requirements)  
**XP Values:** It is based on 4 values:

- **Communication:** It is definitely a key factor to the success of any project as most projects fail because of poor communication. It is achieved by combined and co-located workspaces and development and business spaces, paired development, recurrently changing pair partners, often changing assignments, public status displays, short stand-up meetings and unit tests, demos and oral communication, not documentation.

- **Simplicity:** This refers to developing the simplest product that meets the customer’s needs. It supports delivering the simplest functionality that meets business requirements, designing the simplest software that supports the needed functionality, building for today and not for tomorrow and writing code that is simple to read, comprehend, maintain and amend.

- **Feedback:** It means that developers must obtain and value feedback from the customer, from the system, and from each other. It is provided by aggressive iterative and incremental releases, frequent releases to end users, co-location with end users, automated unit tests, automated functional tests.
• Courage: It refers to making hard decisions that support other principles and practices. Courage is required to do the right thing in the face of opposition and do the practices required to succeed.

**XP Activities:** Coding, Testing, Listening, Designing

**XP Practices:** It is based on following 12 practices:

• Planning Game: It is collaboration between a customer and the developers where iteration planning for next release is performed, customers provide user stories followed by determining budget and schedule estimates.

• Small Releases: It supports the planning game. Working software is delivered in small and frequent releases and is determined in terms of functionality.

• System Metaphor: XP teams develop a common vision of how the program works which is called metaphor. It is the oral architecture of the system which describes how the program works.

• Simple Design: Do as little as needed and provide simplest possible design to get job done. The requirements will change tomorrow, so only do what’s needed to meet today’s requirements.

• Test Driven Development: Extreme programming supports verifying and validating the software throughout the entire project development lifecycle. Developers start by writing test cases first and then write codes as reflected in test requirements followed by user acceptance test and customer approval.

• Refactoring: XP development teams enhance the design of the software all through the whole development lifecycle which is done by refactoring out any duplicate code produced in a coding session. Refactoring is simplified by using automated test cases comprehensively.

• Continuous Integration: New features and changes are incorporated into the system instantly. The development team focuses on continuous integration of the software by verification and validation of the software throughout the product development lifecycle.

• Collective Code Ownership: This suggests that developers own their code and facilitates refactoring.

• Pair Programming: XP Programmers write all production in pairs, two programmers working together at one machine.
• Coding Standards: It follows the concept that the entire code appears to be coded single programmer and distinction on the basis should not be possible.
• On Site Customer: On site customer presence provides quick and continuous feedback to the development team leading to high quality software development.
• 40 Hour Week: XP encourages the development team not to work over 40 hours.

**XP Roles and Responsibilities**

• XP Coach: Guides team to follow XP process
• XP Customer: Writes stories, functional tests and sets implementation priority
• XP Administrator: Setup programmer environment and acts as local administrator
• XP Programmer: Writes tests cases and code
• XP Tracker: Tracks iterations and provides feedback on accuracy of estimates
• XP Tester: Helps customer write, run functional tests and maintains testing tools
• XP Consultant: An external member who guides the team to solve problems

**Limitations**

• XP is not suitable for large, difficult or complex projects.
• It requires great amount of coordination between the programmers while doing pair programming and any small conflict may damage the objective of collective code ownership and hence impact the iterations.
• Development of ‘metaphor’ is required to be shared within team carefully to ensure the common understanding of the terminology.
• Pair programming is a noteworthy practice in XP; in which two developers work on the same machine at the same time and hence it cannot be applied projects with only one developer. Since the testing and coding is done by the same developer, all the probable problems may not be identified as developers test from the same insight the software is created.

**SCRUM**

**Background**
It was first described by Jeff Sutherland, Ken Schwaber, and Mike Beedle in 1996. It focuses on Agile project management technique rather than development aspect of projects. Since Scrum mainly focuses on management skills of both managers and developers, therefore, it can be practiced by any industry (Schwaber et al., 2001).

**Philosophy**
The term ‘Scrum’ is taken from the game ‘Rugby’ where player passes the ball in steps to hit the goal, which is similar as moving, project in iterations to meet the core objective. Scrum is iterative, incremental process to develop any project/product or managing any work. It is a lightweight software development approach that helps to implement most preferred user requirements in small iterations of two to four week sprint cycles. The continuous integration using small sprint reduces the risks and help gaining client confidence. Daily stand-up meeting is very powerful approach to manage and drive the sprint and hence project.

**Scope**
It constitutes of a small team of 7-10 developers and can be scaled to larger numbers.

**Features and Benefits**
**Team size:** Scrum team is usually 5 to 7 members

**Iteration Length:** sprint length is 2-4 weeks

**Phases of Scrum:** Pre-Game (Preparation of product backlog list, effort assessment, high level architectural design); Development (Sprints, analysis, design, delivery) and Post-Game (System testing, integration testing, documentation releases)

**Values:** Supports the values of Respect, Commitment, Focus, Courage, and Openness

**Scrum Techniques:** Team creation, Backlog creation, Project segmentation, Scrum meetings, Burn down charts

**Scrum Meetings:** Sprint planning meeting, daily scrum meeting, sprint review meeting and sprint retrospective meeting

**Scrum Artifacts:**
• Release Backlog: The release backlog includes requirement details and low level estimate as predicted by the team performing the work. These requirements come from the product backlog, are then identified and prioritized for an upcoming sprint or release.

• Sprint Backlog: The sprint backlog assists the development team to calculate the level of effort required to complete a sprint. It is comprised of list of backlog items assigned to a sprint, but not yet completed in common practice.

• Product Backlog: The product backlog is a full list of customer’s desires that includes enhancement requests presently not in the product release.

• Burn Down Chart: The burn down chart is reorganized daily which shows remaining work of a specific sprint. It tracks sprint progress and determines when items must be removed from the sprint backlog and postponed to the next sprint.

Roles and Responsibilities

• Scrum Master: Responsible to facilitate the communication between product owner and team to ensure that Scrum principles are implemented throughout the project.

• Product Owner: Responsible for the success of the product and owns product backlog and manages the project and controls and makes visible the Product Backlog list.

• Scrum Team: Responsible to create a shippable project/product based on client requirement in incremental ways and has authority to decide actions and is self-organizing so as to complete a Sprint.

• Customer: Participates in product backlog items.

• Management: Makes final decision and sets goals and requirements.

Benefits

Scrum fits well into small projects. Some work releases are created and requirements can be prioritized in a well-structured manner.

Limitations

• Usually it works well with small team therefore growing team may require extended coordination.
The project is highly dependent on cohesiveness of the team and the individual commitments of the team members, a minor lack in coordination/communication may cause major impact in the sprint.

Customer is offsite and tight customer collaboration is not possible. Also improved team dynamics enabled by Scrum are not available in one-developer project.

**KANBAN**

**Background**
The Kanban method as formulated by David J. Anderson. It focuses on continuous flow of work instead of sprinting; starting and stopping the delivery of work every 2 to 4 weeks (Anderson *et al.*, 2010).

**Philosophy**
Kanban is a visual process management system that tells what to produce, when to produce it, and how much to produce. It is a technique for managing the creation of products that emphasizes on continuous delivery without overloading the development team. It uses the concept of ‘signboard’ using workflow status (such as TBD, WIP, Done) that provides a comprehensive view of the project and promotes the concept of ‘wide communication’. It advises to reduce the stock level with an objective to reduce the overhead cost and believes in JIT.

**Features and Benefits**
**Kanban Principles:** The 6 principles of Kanban are Visualize the work flow, Limit WIP (Work in Progress), Manage the work flow, Make processes/policies explicit, Implement feedback loops, Improve collaboratively.

**Kanban Practices:** Start with what you do now, Agree to pursue incremental, evolutionary change, Respect the current processes, roles, responsibilities and job titles, and Encourage acts of leadership at all levels

**Roles and Responsibilities**
Kanban does not prescribe any roles and it is dependent on company and team to decide. It suggests reducing the cycle time, for this purpose, the role can be added if it helps in minimizing the cycle time and does not slower the process. Also, it is considered to be unnecessary overhead if the cost of role is higher than the value of enhanced cycle time.

**Limitations**

- A small breakdown in Kanban system’s process can result in entire line shutting down and recovery requires an additional effort.
- The throughput of the Kanban system is not managed instead it comes as a result of controlled WIP and known cycle time.
DYNAMIC SYSTEMS DEVELOPMENT METHOD

Background
Dynamic System Development Method (DSDM) was created through cooperation of a large number of project practitioners who wanted to build quality into Rapid Application Development (RAD) and mainly focuses on early delivery for actual benefits to the company. It was described by Jennifer Stapleton in 1995 (Stapleton et al. 1995).

Philosophy
DSDM is an iterative and incremental methodology that combines the project and product management life cycle into one best process. It was mainly proposed to fill the gaps of Rapid Application Development method by asserting a frame that takes into account of entire software development lifecycle. It is a proven framework for Agile project management that assists in delivering results rapidly and efficiently. It focuses on strategic objective and incremental delivery of real business benefits while controlling budget, schedule, risk and quality.

Scope
DSDM is not suitable for all projects. In particular, systems that are real-time, safety critical, or have well defined requirements are not suited to its use. DSDM is especially suited to projects with changing requirements. It is critical that these requirements be capable of being prioritized.

Features and Benefits
Team Size and Iteration Length: DSDM is not so much a method as it is a framework. Because of DSDM’s framework nature, it does not specifically address team size and exact iteration lengths. Team size varies from two to six people but there may be many teams in a project.
DSDM Phases: The DSDM lifecycle has 6 stages: Pre-project, Feasibility Study, Business Study, Functional Model Iteration, Design and Build Iteration, Implementation, and Post-project.
Techniques: Time boxing, MoSCoW, Prototyping, Testing, Workshop, and Modelling. MoSCoW is famous prioritization technique, it indicates ‘Must have’, ‘Should have’, ‘Could have’ and ‘Would nice to have’ while prioritizing the features. DSDM is heavier than XP and Scrum as well as competent in terms of budget and time. It provides a technique independent process and is flexible in terms of requirement advancement.
**DSDM Principles:** DSDM relies on nine principles:

- Involve active users
- Empower teams to make their own decision
- Prefer frequent delivery releases
- Meeting the business need is the main criteria for releases
- Plan for iterative development
- Any change during development can be inverted
- The most high level requirements should be unchangeable
- Testing should occur throughout project lifecycle
- All stakeholders must cooperate and communicate

**Roles and Responsibilities**

- **Executive Sponsor:** Responsible for making decisions in assigning appropriate funds and resources
- **Visionary:** Responsible for initialising the project by making sure that important requirements are identified early and supervising and keeping the development process in the right track
- **Ambassador:** Responsible to make sure that the team receives customer feedbacks during the entire development process
- **Advisor:** Responsible in representing viewpoint and brings daily project knowledge
- **Project Manager:** Responsible to manage the project
- **Technical Co-ordinator:** Responsible in designing the system architecture and controls the technical quality in the project
- **Team Leader:** Leads and makes sure that the team works efficiently as a whole
- **Solution Developer:** Responsible to interpret the system requirements and model it
- **Solution Tester:** Responsible to check the technical correctness by testing
- **Scribe:** Responsible to gather and record the requirements, agreements, and decisions made in every workshop
- **Facilitator:** Responsible in managing the workshops progress
- **Specialist Roles:** Business Architect, Quality Manager, System Integrator
Limitations

- DSDM require active user involvement that might not be possible in every project
- The other drawback in adopting DSDM is that it can be restrictive and not easy to work with other Agile development software methods due to its nine principles.

FEATURE DRIVEN DEVELOPMENT

Background

The concept of Feature Driven Development (FDD) started in the late 1999 from collaboration between Jeff DeLuca and Peter Coad and was later described by Palmer and Felsing in 2002. (Palmer and Felsing, 2001). It focuses on the domain model and consists of five activities: Develop an overall model, Build a list of features, Plan by feature, Design by feature and Build by feature.

Philosophy

The main focus of feature driven development is on the design and building phases. It provides emphasis to quality aspects throughout the project lifecycle and includes recurrent and substantial deliveries, along with monitoring the project progress. It is simple to understand but powerful approach to build the product or solutions.

Scope

FDD is limited to small to medium sized teams (4 - 20 people). The methodology deals with uncertain requirements and center on Object Oriented modeling.

Features and Benefits

Team Size: The size of team fluctuates with the complexity of the feature at hand
Iteration Length: Up to two weeks
FDD Phases: The five phases are Develop overall model; Build the Feature, Plan by Feature, Design by Feature and Build by Feature
FDD Practices: Domain Object Modeling, Developing by Feature, Individual Class (Code) Ownership, Feature Teams, Inspections, Configuration Management, Regular Builds, Visibility of progress and results

FDD Activities: Develop Overall Model, Build Feature List, Plan By Feature, Design By Feature, Build By Feature, Milestones

FDD Values: A method for building a system is crucial for larger projects. For this purpose, a simple, well defined process and logical process step works best. These processes move to background so that development team can focus on feature driven lifecycle.

Roles and Responsibilities
- Project Manager: Responsible for all administrative aspects of the project
- Chief Architect: Responsible for the overall design of the system
- Development Manager: Responsible for daily software development activities
- Chief Programmer: Responsible for on-going design and development activities for one or more Feature Sets.
- Class Owner: Responsible for designing, coding, testing, and documenting the features as they are implemented and works under the direction of chief programmer.
- Domain Expert: Responsible for defining the required features of the system.
- Tester: Responsible for verifying that each feature performs as defined.
- Deploy Manager: Responsible for actual code deployment to various environments.
- Technical Writer: Responsible for creating and maintaining all the documentation.

Limitations
The FDD prescription does not specifically address project criticality.

LEAN SOFTWARE DEVELOPMENT

Background
Lean Software Development (LSD) was proposed by Mary Poppendieck and Tom Poppendieck. It was adapted from Lean manufacturing of TOYOTA production system and Bob Charette’s Lean development in the 1980s. It focuses on the project management aspects and does not point
any technical practice but does assimilate well with other Agile techniques that focus more on the technical aspects of software development (Poppendieck et al., 2005).

**Philosophy**

Lean Software Development is an iterative methodology that focuses on reducing waste and optimizing the entire process to achieve the maximum possible gain. Lean has a good record in manufacturing industry and has gained recognition in software development industry in recent years. It integrates well with the concept of six sigma.

**Scope**

Any software development project where there is need for radical change and the major scope focuses at company CEOs. No team size specifications are laid out because LSD is more of a software development management philosophy than a methodology.

**Features and Benefits**

**Team Size and Iteration Length**: Since LSD is more of a management philosophy than a development process, team size and iteration length are not directly addressed. 

**LSD Principles**: It seven Lean principles, the methodology revolves around these principles, and all other aspects of Lean are designed to reinforce them.

The seven principles are:

- **Eliminate Waste**: Remove anything that does not add value to the customer
- **Build Quality In**: Validate and re-validate all assumptions throughout the development process and discard all the practices that no longer have value.
- **Create Knowledge**: Use short iterative cycles to provide rapid, constant feedback to make sure the right things are being focused on.
- **Defer Commitment**: Do not judge until enough is known to make the decision.
- **Deliver Fast**: Reduce the time it takes to list a business problem and deliver a working software fast.
- **Respect People**: Empower the team to succeed.
• Optimize the Whole: Use cross functional teams to keep from missing important, possibly critical aspects of the problem and of the system designed to solve it.

 Usually only customer provides business requirement to the team and play vital role. The prime objective is to eliminate the waste and optimize the entire process to achieve the maximum possible gain. Integrates well with the concept of six sigma.

 Roles and Responsibilities
 No specific mention of roles and responsibilities are laid out except that LD is aimed at CEOs before it can be implemented in the organization.

 Limitations
 • The project is highly dependent on cohesiveness and the individual commitments of the team members therefore team building is critical factor.
 • An inapt involvement from appropriate business analyst could result in scope creep.
AGILE MODELING

Background
Agile Modeling (AM) was proposed by Scott Ambler in 2002 (Ambler et al., 2002). It focuses only on documentation and modeling. It can be used with any software development process as it is not a complete software development method.

Philosophy
Agile Modeling (AM) is an approach to perform and adapt modeling practices using an Agile principles. It is a practice based method for effective modeling and documentation of software based systems. At a high level, it is a set of best practices and at a more detailed level, it is a set of values, principles, and practices for modeling software that can be applied on a software development project in an efficiently. AM explicitly includes part of the Disciplined Agile Delivery (DAD) framework and provides enhancement to other Agile techniques such as XP, Scrum and RUP. The main objective of AM is lower the amount of models and documentation as much as possible. The cultural challenges are addressed by encouraging communication and by organizing team structures.

Scope
No specific team size is mentioned but the methodology aims for small teams. The AMDD framework can be combined with all non-modeling Agile methodologies.

Features and Benefits
Team Size and Iteration Length: The size of the team and iteration lengths usually depends on the development process being used.
Values: communication, simplicity, feedback, courage and humility
Goals: The three main objectives are to identify and demonstrate how to put into practice a collection of values, principles and practices for efficient modelling; to address the challenge on how to implement modeling techniques on Agile processes; and to address how to implement efficient modeling techniques individually of the software process in use.

Roles and Responsibilities
AMDD teams are expected to come from developers and project stakeholders. Teams should be composed of self-motivated hard working developers. The modeling must be done in teams where everyone must participate.

Limitations

- AM disciplines can be complicated to implement, specially on large teams without sufficient tooling support
- AM may be difficult to use where team members are unable to share and collaborate on models
- When modeling skills are weak or lacking

ADAPTIVE SOFTWARE DEVELOPMENT

Background

It was described by Jim Highsmith and Sam Bayer in 2000 (Highsmith et al., 2000). Adaptive Software Development (ASD) is part of rapid application development and focuses on rapid creation and evolution of software systems.

Philosophy

ASD is a method for creating and developing software systems. It recommends solutions for developing large and complex systems through continuous prototyping, incremental and iterative development. It involves product initiation, adaptive cycle planning, concurrent feature development, quality review, and final quality assurance and release. Usually customer does not have all the requirements in the beginning and act as a member with the concept of progressive elaboration.

Scope

Adaptive software development is basically a management philosophy and limited to project management activities. The restrictions of ASD begins when it comes to team sizes depending on the project size, but like any other Agile method, the level of agility reduces as the team gets larger.

Features and Benefits
Phases: Speculate, collaborate and learn.


Roles and Responsibilities
The primary roles of ASD are Executive Sponsor, Participants, Developer and Customer

CRYSTAL

Background
It was developed by Alistair Cockburn in 2001 (Cockburn et al., 2001). It focuses more on people rather than process.

Philosophy
Crystal methods are collection of Agile methods that selects the most suitable one and tailoring them for each individual project based on project complexity and team size. Larger projects are likely to ask for more coordination and heavier methods than smaller ones. It is demonstrated in four colours: Red for extreme large size, Orange for large size, Yellow for medium size and Clear is applied to small teams working on developing software that are not life-critical.

Scope
The Crystal family of methodologies is essentially a project management philosophy that defines projects according to team sizes. It is rather difficult to spell out the scope of Crystal because the methodology provides a basis for selecting and tuning other methodologies.

Features and Benefits
Team size: The Crystal Family contains any team size. In Crystal Clear the team size is up to six developers. In Crystal Orange the team size is from ten up to forty developers. Crystal methodologies are not suitable for life–critical systems.

Iteration Length: For large and crucial projects, up to four 4 months
Properties: Frequent delivery, reflection enhancement, close communication, personal safety, focus, easy access to expert customers, technical environment with automated tests, configuration management and frequent integration.

Strategies: Exploratory 360°, Early victory, Walking skeleton, Incremental, re-architecture, Information radiators.

Techniques: Methodology shaping, Reflection workshop, Blitz planning, Delphi estimation using expertise ranking, Daily stand-up meetings, Essential interaction design, Process miniature, Side-by-side programming, Burn charts.

Roles and Responsibilities
Crystal Clear has only one team and Crystal Orange has several teams.

- **Sponsor:** Responsible to finance the project and delivers the mission statement
- **Senior Designer:** Responsible to maintain team structure, implement methodology and designs the system
- **Designer/Programmer:** Responsible to create screen drafts, design sketches and notes, common object model, source code, packaged system, migration code, and test cases
- **User:** Responsible in helping with use case and screen drafts.
- **Business Expert:** May come from the sponsor, the user or the senior designer.
- **Coordinator/Tester/Writer:** May come from the designers.
- **Crystal Orange** has the following additional roles arranged into teams: system planning, project mentoring, architecture, technology, functions, infrastructure and external test teams.

Limitations
Crystal supports 4 basic criticalities: failure resulting in loss of comfort, discretionary money, essential money, and life.

AGILE UNIFIED PROCESS

Background
Agile Unified Process (AUP) is a simplified version of the IBM Rational Unified Process (RUP) developed by Scott Ambler and was introduced in 2005 and further revised in 2006 (Ambler et. al., 2006). It mainly focuses on real-time and web-based development.

**Philosophy**

Agile Unified Process explicates a simple approach in developing software using Agile principles and practices in addition to RUP features. In 2012, the AUP was superseded by Disciplined Agile Delivery (DAD) and since then work has ceased on evolving AUP.

**Scope**

The AUP executes Agile principles, such as test driven development, Agile Modeling, Agile change management, and database refactoring to advance productivity.

**Features and Benefits**

**AUP Philosophies:** AUP assumes that staffs know what they're doing. The team member does not prefer reading process documentation but do expect frequent high level guidance. The AUP product offers links to many details but doesn't force them.

- Simplicity: Everything is explained concisely in few pages
- Agility: It follows the Agile software development values and principles
- Focus: It focuses on high level activities of a project.
- Tool independence: Any toolset can be used with the AUP and it is recommended to use the one which is best suited for the job

**AUP Disciplines:** Disciplines are carried out in an iterative method by defining the activities that development team executes to build, validate, and deliver working software to meet stakeholder's requirement. Each iteration consists of following seven disciplines:

- Model: The objective is to comprehend the business of the organization, the problem domain deal with by the project, and identify a feasible solution to address the problem area.
- Implementation: The aim is to change the model(s) into executable code and carry out a basic level of testing, particularly unit testing.
• Test: The aim is to carry out an objective evaluation to ascertain quality. This includes finding errors, validating the system so that it functions as per the design, and verifying that the requirements that were outlined, are met.

• Deployment: The purpose is to chart out a delivery plan of the system and to implement the plan to make the system accessible to end users.

• Configuration Management: The objective is to manage access to project components including tracking component versions over a period and controlling and managing alterations to them.

• Project Management: The aim is to direct the activities that take place within the project. This includes managing risks, directing people (assigning tasks, tracking progress, etc.), and coordinating with people and systems outside the scope of the project to be sure that it is delivered within the stipulated time and budget.

• Environment: The objective is to support the rest of the effort by making sure that the proper procedure, direction (standards and guidelines), and kits (hardware, software, etc.) are accessible to the team as required.

For each discipline, AUP defines sets of Artifacts (work products), Activities (units of work on the artifacts); and roles (tasks taken on by development team members).

**AUP Releases:** The Agile Unified Process distinguishes between two types of iterations: a development release iteration which results in a deployment to the quality-assurance and/or demo area, and a production release iteration which results in a deployment to the production area and is a significant refinement to the RUP.

**AUP Phases:** The overall development cycle consists of following four phases:

• Phase 1: Inception: The aim is to recognize the original scope of the project, a probable architecture for the system, and to acquire original project funding and stakeholder acceptance.

• Phase 2: Elaboration: The aim is to establish the architecture of the system.

• Phase 3: Construction: The aim is to produce working software on a frequent, incremental basis to meet stakeholder's requirements priority wise.
- **Phase 4: Transition:** The aim is to validate and implement the working software into production environment. Every phase can be in addition broken down into Iterations. Iteration is a full development loop that result in a release of an executable increment to the software.

**Limitations**
- Modeling may risk agility, if restrictions are not strictly observed
- Tackling model inconsistency is not explicitly addressed.
- The AUP isn't for everyone. It adopts XP principles and practices yet retaining some of the formality of RUP. XP is preferred by developers for light weight process and RUP is preferred by management for well defined process and due to large number of artifacts it offers.

**ADVANTAGES OF AGILE SOFTWARE DEVELOPMENT**

Agile principles and values are not only beneficial to software development teams but also provide business benefits to the customer and clients. In addition, Agile practices helps in addressing common project drawbacks, such as, scope creep, budget control and schedule forecast. The main advantages of adopting Agile principles in software development projects are explained as follows:

1. **Rapid, Iterative and Incremental Delivery:** The delivery of project is divided into small functional releases that checks the functionality and gets early and frequent customer feedback in order to control any potential risk or threat. The releases are delivered quickly and frequently on a fixed schedule iterations of 1-4 weeks each, with a high level of predictability. The project plans, requirements, designs, code and tests are produced in the beginning and reorganized incrementally as required to adapt the project changes. This helps in checking and monitoring the software functionality progress on regular basis rather than at end of lengthy milestones.

2. **Increased Performance:** The daily stand-up meetings are conducted to allow exchange of valuable information among development team members and to ensure continuous progress.
Frequent communication helps in sharing knowledge which further improves team productivity and followed by good return on investment.

3. **Flexibility of Design:** The flexibility of design depends on the projects development process and is defined as ability to change directions quickly. The main feature of Agile approach is to adapt to changing requirements quickly which enables the design to be made flexible in order to handle the changes with ease.

4. **Adaptive to the changing environment:** Using an Agile approach, the working software is developed and delivered in several iterations in order to get frequent customer feedback after every iteration. Agile principles encourage and implement any change requirement from the user at any stage of product development to upgrade the software.

5. **Reduces development risks:** The working software is provided to users in small and incremental releases. Based on the customer’s feedback, developers are notified of any potential problem that may crop up at the later stages of the development and helps fixing it immediately in order to reduce the risk of development.

6. **Working Software:** The Agile practices focuses on delivering working software frequently which ensures greater reliability and opportunity to incorporate customers feedback in order to build best product as per users need and requirements.

7. **Ensures customer satisfaction:** The Agile values and principles promote active user involvement throughout the software development lifecycle. The small releases are provided to the customer for use and further development to the software is done based upon their feedbacks. This ongoing process ensures high quality delivery of the final product and customer satisfaction.

8. **Avoids over production:** The traditional system requirement document is still built whether many features required or not. Using Agile principles helps create best product by building it for now (if required) and not later.
9. **Improvement in Quality:** The quality of software is produced by developing features, conducting continuous testing and getting frequent customer feedback throughout the release of each iteration. This helps in finding, fixing the errors fast and identifying potential problems early that leads to higher code reuse and better quality.

10. **Least documentation:** The Agile practices support least documentation that is required and internal design of the software is usually not documented. The main contents of the documents are list of product features and function, duration for each iteration, the product development and deliverable duration.

11. **Fault Detection:** The continuous and integration testing principles of Agile enforces the delivery of high quality bug free software. As testing is performed during each iteration, error and defects are identified earlier and are fixed instantaneously before it turns into a risk.

12. **Best Practices:** The development team can produce and deliver high quality softwares provided they incorporate core principles and practices of Agile methodologies. The Agile philosophy encourages 'architecture killers', which believes to fail early than later in the project that also saves budget and time.

**DISADVANTAGES OF AGILE SOFTWARE DEVELOPMENT**

Besides many advantages of using Agile methods and like traditional methods, the Agile methods also have some limitations and have been criticized by researchers and practitioners, as explained in detail:

1. **Not suitable for large projects:** Agile methods are considered to be more suitable to small projects with small teams and does not scale well to large projects as number of iterations are required to build the desired feature and functionality of a project. If Agile methods are applied on larger projects, it may lead high opportunity cost.
2. **Customer Interaction:** Agile method requires active user participation and higher customer interaction with the project team during the development of software product that ensures high quality delivery of right product. However, in practice, these principles are difficult to be followed as it demands a huge commitment from the customer during the product development lifecycle.

3. **Insufficient and Unclear Requirements:** During the initial stage of project, the user requirements are insufficient and unclear. The requirements are clarified and specified during the development phase just in time and are not documented in detail due to the timeliness of conversations. However, it helps the project to start building the product but if the client is not clear about the product requirements, features and final outcome, the project can easily get taken off track.

4. **Changing Requirements:** The user requirements may change throughout the product development lifecycle. The flexibility to make desired changes ensures delivery of the right product. However, this principle of Agile creates the risk of never ending projects and there is much less predictability during of the project lifecycle final product delivery requirement. Without sufficient and clear vision, it gets difficult to control cost; schedule and scope creep of a project.

5. **Difficulty in Integration Testing:** Integrated testing is performed throughout the product development lifecycle which ensures the high quality throughout the project without the need for an extensive testing at the end of the project. This increases the resource cost of the project, as this practice of Agile demands testers throughout the product lifecycle.

6. **Frequent Delivery:** Adopting an Agile approach promotes frequent delivery of product, followed by User Acceptance Testing which requires the product owner and testers to be available throughout the project development lifecycle for testing features as they are developed. Though, this process ensures high quality product delivery but consumes lot of time and these endless iterations can be mentally exhausting, therefore it’s imperative to determine a sustainable pace for the team.
7. **Lack of Documentation:** The use of Agile practices supports less documentation that saves lot of development time but this is huge drawback for the developers. Because of the tight project deadlines, it is not easy to maintain the detail documentation as internal design changes frequently based upon users changing requirements after every iteration. Hence, it is very difficult for new developers to understand the work and develop with this limited documentation.

8. **More helpful for management than developer:** Agile methods facilitate management to make decisions regarding setting goals and deadlines for software developers to work on. However, it difficult for the baseline developers to cope up with the ever changing requirements and design. In addition, it increases the management overhead as it requires strong teamwork and project managers are expected to be involved during in the development of entire product.

9. **Culture and Co-located Teams:** Quite often, software development teams are located in different parts of the world and having face to face communication is almost impossible. To overcome this problem, most developers have to use video conferencing tool to deliver high quality product which can be sometimes challenging.

10. **Experienced Resources:** Agile approach allows only senior programmers to take decisions required during the development process and does not allow novice developers to make decisions unless combined with experienced resources.

11. **Traditional Waterfall Development mindset:** The organizations currently practicing waterfall development model are reluctant to adopt and implement Agile methods. Agile principles emphasises more on development rather than design and focuses on processes for getting requirements, developing code and does not focus on product design which can be sometimes challenging. Agile is also not suitable for maintenance due to less documentation for the systems.
12. **Unfamiliarity with Agile:** To get the benefits of implementing Agile principles and values in software development projects, there are some assumptions that needs to applied in order to get better results, such as frequent and small releases, embracing changing requirements, close collaboration between the clients and the development team.
1.2 Mobile Application Development

The mobile software development is the process by which applications are developed for handheld devices, such as mobile phones and tablets. These applications can be either pre-installed on phones at the time of manufacture or can be downloaded from various mobile software distribution platforms. The development of mobile applications is currently challenging due to rapidly changing business requirements and technical constraints associated with mobile systems. Development teams therefore face the challenge of a dynamic environment, with frequent modifications in customer needs and expectations.

The rapid growth of mobile computing platform has almost outpaced the software engineering processes tailored to mobile application development with significant modifications. However, at the moment, there is limited knowledge about the suitability of different software practices for the development of mobile applications. Agile software development practice have caught the attention of software development teams and software engineering researchers worldwide during the last decade but scientific research and published outcomes still remains quite scarce.

In recent years, the mobile industry has witnessed rapid expansion; the potential number of different mobile applications is almost unlimited. With the increasing popularity and demand for mobile applications, there has been significant increase in number of projects for mobile application development services. As of result of this, the quality and quantity of the mobile applications has introduced new concerns in computer science and industry.

Presently mobile phones are gaining a new dimension with the emergence of smart phones and tablets. Mobile application development has been steadily growing, both in terms of revenues and jobs created. There is continuous increase in number of mobile applications developed and downloaded every year and the sudden emergence of mobile devices has made new computing platform financially beneficial for not only entrepreneurs but also for independent software developers.
According to the VisionMobile and Plum Consulting survey (2013) analyst report estimates there are 529,000 direct App Economy jobs within the EU 28 members, 60% of which are mobile app developers.

Mobile Application Types and Categories

To be successful in mobile application development (MAD), it is imperative to have a profound understanding of what mobile applications really are and how they differ from desktop applications that users also interact with. ‘Mobile Application Types’ are generally confused with ‘Mobile Application Categories’. Table 1 and Table 2 describes various ‘types’ and ‘categories’ implicated in the development of mobile apps with suitable examples so as to clearly differentiate between types and categories (Abhineet S., 2012).

Table 1: Mobile Application Types

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Browser Access Apps</td>
<td>Apps are not installed in the device and can be accessed through native browser by hitting the URL of the web. The device memory size is not imperative as the app data is not stored in the device. It is completely dependent on the quality of the browser. For example, m.yahoo.com, google.com.</td>
</tr>
<tr>
<td>2 Native Apps</td>
<td>Apps are installed in the device. They do not need any data transfer to the server and works in the device without network as the data about the app is stored in the device itself. For example, Notes and Reminder in iPhones.</td>
</tr>
<tr>
<td>3 Hybrid Apps (Web)</td>
<td>Apps are installed in the device and always require internet connection to run and function. For example, Social Networking Apps (Facebook, Twitter), Instant Messengers (Skype), E-Commerce (Flipkart), Internet Speed Testing (Speedtest).</td>
</tr>
<tr>
<td>4 Hybrid Apps (Mixed)</td>
<td>Apps are installed in the device and may or may not require internet connection to run and function. For example, Medical apps and few games in that can be played alone, offline and go online too for playing with multiple players.</td>
</tr>
</tbody>
</table>

The various mobile applications that are available on any apps store these days can be categorized into the following categories:

Table 2: Mobile Application Categories

<table>
<thead>
<tr>
<th>Categories</th>
<th>Description</th>
</tr>
</thead>
</table>
1.3 Using Agile approach in Mobile Application Development

For mobile development teams looking to introduce a lightweight development process or scale back more bureaucratic processes, Agile software development offers tremendous opportunities and value. Progress in mobile computer technology and the rapid escalation of wireless networks in quality and quantity has brought in new applications and concerns in computer science and industry.

The distinctive requirements and constraints associated with mobile systems have brought new challenges to software development for such environments, as it requires extensive improvements to conventional systems development techniques in order to fulfil the special needs of this field.

The Agile approach is seen as a natural fit for mobile application development and studies carried out for the application of the Agile development approach to mobile application development indicates the need for software development processes tailored to suite the mobile application requirements. There are many reasons why Agile Software Development suites Mobile Application Development these include small teams, short deadlines, importance of usability, fast delivery and less complexity.

1.4 Motivation
Conventional software development methods have gradually been replaced by lightweight Agile software development methods since the mid-1990s. This phenomenon is mainly due to the shortcomings of the conventional method including a slow adaptation to rapidly changing business requirements, and a tendency to be over budget and behind schedule. The mobile telecommunications industry is comprised of a highly competitive, uncertain and dynamic environment. The potential number of different mobile applications is virtually unlimited. With the increasing popularity and demand for mobile applications, there has been significant increase in number of projects for mobile application development services.

Agile innovations offer a solution for mobile application and service developers who are in need of high quality development processes. There is a great necessity to explore various Agile methodologies for the development of mobile applications because Agile approach is a natural fit for the development of mobile applications as traits observed in mobile software development maps perfectly to Agile themes.

1) **Proposed Agile models for mobile software development:** There are five methodologies, Mobile-D, MASAM, HME, SLeSS and Scrum as proposed by researchers for the development of mobile applications using various Agile approaches. The review of the literature on above mentioned methodologies brings forth the following observations which indicate substantial scope for further research in the domain:

   a. **Mobile D:** Mobile-D was the first attempt to incorporate Agile practices for the development of mobile applications and was introduced as a development methodology inspired on Extreme Programming, Crystal Methodologies and Rational Unified Process (Abrahamsson et al., 2004). Mobile D software development process was highlighted as a promising technology, however the description provided for the methodology was cursory and incomplete. In spite of being a pioneering study done in 2004, it lacked further improvements using hybrid Agile techniques and experimental validation in a real organization.

   b. **HME:** Rahimian et al. (2007) proposed the hybrid method engineering development framework that takes into account most of the issues identified in the field. However, the
methodology is still at a high-level, and no specific tasks for the identified stages have been provided. The future work may include performing further iterations to obtain lower-level tasks in the process. However, in its current state the methodology is more at theoretical level with Mobile-D.

c. **MASAM**: Jeong *et al.* (2008) proposed this methodology included in Mobile-D, with a minor original contribution in the methodological area. The locale conditions of the tool made it unlikely to be widely extended. The MASAM methodology claims to be supportive for small companies focused on the development of mobile software applications. However the methodology lacks a case study of an actual implementation in a real-world environment to appreciate the results.

d. **Scrum**: Scharff and Verma, (2010) demonstrated the use of Scrum for the development of mobile applications. This study focuses primary on project management but fails to implement the engineering practices which is imperative while developing a mobile application.

e. **SLeSS**: Cunha *et al.* (2011) proposed combination of Scrum with Lean Six Sigma for the development of mobile applications. The focus of their study was more on project management aspect rather than engineering practices. The proposed study included a case study of an implementation in a real-world environment; however the utilization report was not reported in their study.

2) **Other Agile methodologies for mobile software development**: Beside the use of above proposed methods, other Agile approaches can also be investigated and integrated for the development of mobile applications. For example, Scrum, is a project management methodology, extreme programming is engineering discipline, Feature Driven Development (FDD) is a requirement methodology that provides excellent practical advice if the group typically works on features, Dynamic Systems Development Method (DSDM) which has fundamental practices and emphasizes heavy prototyping. The combination of Scrum, XP, Lean and Kanban has proven to be beneficial in the development of other software products.
It was proposed that mixing of these approaches can be applied to mobile software projects (Kannan et al., 2011; Holler et al., 2011; Corral et al., 2013; Morris et al., 2010; Murphy et al., 2011; Habermas et al., 2013; Wasserman et al., 2010; Dehlinger et al., 2011).

3) **Interview with mobile development community:** Beside the use of proposed methods, there is necessity to explore other Agile approaches that can also be investigated and integrated with the development of mobile applications. This can be done by means of industrial surveys, interviews with mobile software developers, mobile project managers, concrete discussion with Agile experts and other experimental studies.

4) **Limited study concerned on important aspect of MAD:** Despite of the identified business opportunity, a very few scientific publications can be found which address the specific problems that the development organizations are facing while developing software’s for mobile devices. There is scarcity of research studies that demonstrate the characteristics of mobile app development that makes it different from traditional software development; Mobile Software Development Lifecycle (MSDLC) process; issues and challenges faced by mobile developers during application development process; best practices that should be considered while developing a mobile app; and advantages, disadvantages & concerns regarding the use of Agile in the development of mobile applications.

Agile software development practice have caught the attention of software development teams and software engineering researchers worldwide during the last decade but scientific research and published outcomes still remains quite scarce in the following listed areas (Figure 1):
5) **Timeliness of the methodologies:** During the initial stages of mobile-specific development process, Agile approach was considered as the best fit for mobile development. Nevertheless, in those days the mobile business and development and execution environment were different from the current one. Abrahamson et al. (2005) for the first time demonstrated the mapping between the Agile home ground themes and the characteristics of the mobile software. However, the current applicability of this mapping is controversial and invites to conduct an up-to-date discussion since more than one decade of evolution in the mobile domain (software, hardware and business models) has brought significant advancements in this area.

Few specific differences of the current status of the mobile domain are provided below:

- While hundreds of new mobile models are still released each year, mobile developers also have well settled operating systems that have development kits (SDK) and APIs that facilitate the interaction with new device models.
• The current status of the mobile-specific assurance practices provides software engineers with a solid body of knowledge to combine several development practices to decide a specific approach that includes Agile practices, plan-based methodologies, stringent product-oriented quality inspections, software metrics, and design patterns or implementation techniques.

• Beside the use of above proposed methods, other Agile approaches needs to be investigated and integrated for the development of mobile applications, such as, Scrum, XP, Lean, Kanban, DSDM as well as practices from traditional waterfall development methods. This will reduce the time-to-market, effort, cost of application development and deployment, and improve overall quality of mobile development project.

Therefore the proposed work will ascertain a significant and distinctive contribution towards better understanding and advancement of the existing process of mobile application development. It is strongly believed that the results of this study will significantly contribute towards the knowledge domain of mobile software engineering.