ISSN:0975-9646

A Systematic Study on Agile Software Development Methodologies and Practices

Harleen K. Flora[1], Swati V. Chande[2]

[1]*Research Scholar, The IIS University, Jaipur, India*
[2]*Professor, International School of Informatics and Management, Jaipur, India*

Abstract **- Software engineering techniques have been employed for many years to create software products. The selections of appropriate software development methodologies for a given project, and tailoring the methodologies to a specific requirement have been a challenge since the establishment of software development as a discipline. In the late 1990's, the general trend in software development techniques has changed from traditional waterfall approaches to more iterative incremental development approaches with different combination of old concepts, new concepts, and metamorphosed old concepts. Nowadays, the aim of most software companies is to produce software in short time period with minimal costs, and within unstable, changing environments that inspired the birth of Agile.**

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. Every agile approach has its own development cycle that results in technological, managerial and environmental changes in the software companies. This paper explains the values and principles of ten agile practices that are becoming more and more dominant in the software development industry. Agile processes are not always beneficial, they have some limitations as well, and this paper also discusses the advantages and disadvantages of Agile processes.

Keywords **- Agile Software Development, Agile Methodologies, AUP, AM, AUP, DSDM, FDD, XP, Scrum, Lean, Kanban, Crystal.**

I. INTRODUCTION

The Waterfall Model is a sequential development approach, in which development flows downwards through the phases of requirements analysis, design, implementation, testing and maintenance.

In waterfall model, project is divided into sequential phases; emphasis is on planning, time schedules, target dates, budgets and implementation of an entire system at one time. Tight control is maintained over the life of the project via extensive written documentation, formal reviews, and approval/signoff by the user and information technology management occurring at the end of most phases 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. The Waterfall model is a traditional engineering approach applied to software engineering. It has been widely blamed for several large-scale government projects running over budget, over time and sometimes failing to deliver on requirements due to the Big Design Up Front approach.

1.1 Agile Software Development

Agile Development Model is based on iterative and incremental development approach in a highly collaborative manner to produce high quality software in a cost effective and timely manner which allows the project to adapt the changes quickly. Agile methodologies stresses on delivering the smallest working piece of functionality as early as possible and constantly improving it and adding additional functionality throughout the project lifecycle.

Agile helps in minimizing and mitigating the overall risk, and allows the project to adapt to changes quickly and does not require a requirements freeze upfront unlike waterfall model. Work is carried out in iterations, which typically last one to six weeks. The industry mostly opts for agile development process as it is highly result oriented. Agile methods emphasize effective communication over written documents. The Project requirements are well documented upfront. Then, depending upon business priority, these features are assigned to releases, which are tied to iterations. Agile methods emphasize working software as the primary 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 ROI.

Although, there are many success stories of using Agile in software development project in last decade but knowledge of implementing these practices in a particular project is very scared. Therefore, this publication 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. This paper also provides the benefits and limitations of Agile Software Development and recommends when to use them.

1.2 Agile Manifesto

In recent years, with the rising competence of software market, researchers are seeking more flexible methods that can be used to adjust to dynamic situations where software system requirements are changing over time. In 2001, the Agile manifesto [1], established the approach now known as agile software development process created by 17 influential figures, a guiding force for Agile practitioners which details four core values for enabling high-performance, efficiency and outputs:

1. Individuals and their interactions, over processes and tools.
2. Delivering working software, over comprehensive documentation.
3. Customer collaboration, over contract negotiation.
4. Responding to change, over following a plan.

The items on the right have value, however, the items on the left define the agile philosophy. Exploring each of these values will aid in gaining knowledge of the agile process philosophy while exposing how applying the philosophy to defined methods will enhance software development aligning it with today's volatile markets.

1.3 Agile Principles

The Agile Alliance also documented the principles they follow that underlie their manifesto [2]. As such the agile methods are principle-based, rather than rule-based. Rather than have predefined rules regarding the roles, relationships, and activities, the team and manager are guided by these principles:

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity, the art of maximizing the amount of work not done is essential.
11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

II. AGILE METHODOLOGIES

Several Agile techniques have been proposed and used by researchers in difference domains. This section identifies some of the well-known existing agile software development methods and their objectives. Following various Agile methodologies share common principles, but differ in practices:

2.1 Extreme Programming (XP)
2.2 Scrum
2.3 Lean Software Development (LSD)
2.4 Kanban
2.5 Adaptive Software Development (ASD)
2.6 Feature Driven Development (FDD)
2.7 Dynamic System Development Method (DSDM)
2.8 Agile Modeling (AM)
2.9 Crystal
2.10 Agile Unified Process (AUP)

2.1 EXTREME PROGRAMMING (XP)

A. Background

It was introduced in 1998 by Kent Beck, Ron Jeffries, Ward Cunnigham based on experience from C3 project [3].

B. Philosophy and Scope

Extreme Programming (XP) is a well-known and a light weight discipline of software development that focuses on engineering practices. XP aims at enabling successful software development despite unclear or constantly 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 consists of gathering informal requirements from on-site customers, organizing teams of pair programmers, developing simple designs, continuous refactoring, continuous integration and testing; and advocates frequent releases in short development cycles that improves productivity as well introduces checkpoints where new customer requirements can be embraced [4,5,6]. XP is best suited for projects that require collocated teams of small to medium size team. On the project side XP is meant for projects where the requirements are unstable and unpredictable.

C. 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 (write story for current iteration), Iteration Planning (Prioritize Stories, effort and resource estimates), Iteration to release (Analysis, design, coding, testing), Production (Rigors testing), Maintenance (Customer supports, release for customer use), Death Phase (No more requirements).
XP Values: It is based on 4 values:

a. *Communication:* It is definitely a key factor to the success of any project as most projects fail because of poor communication. It is accomplished by collaborative workspaces, co-location of development and business space, paired development, frequently changing pair partners, frequently changing assignments, public status displays, short stand-up meetings and unit tests, demos and oral communication, not documentation.
b. *Simplicity:* It means to develop the simplest product that meets the customer's needs. It encourages delivering the simplest functionality that meets business needs, designing the simplest software that supports the needed functionality, building for today and not for tomorrow and writing code that is easy to read, understands, maintains and modify.
c. *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.

d. *Courage:* It means to be prepared to make 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:

a. *Planning Game:* It is collaboration between a customer and the developers where planning for the upcoming iteration is done, user stories are provided by the customer, technical persons determine schedules, estimates, costs, etc.
b. *Small Releases:* It supports the planning game. Small releases are in terms of functionality and less functionality means releases happen more frequently.
c. *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.
d. *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.
e. *Test Driven Development:* XP teams focus on validation of the software at all times. Programmers develop software by writing tests first, and then software that fulfills the requirements reflected in the tests. Customers provide acceptance tests that enable them to be certain that the features they need are provided.
f. *Refactoring:* XP teams improve the design of the system throughout the entire development. This is done by refactoring out any duplicate code generated in a coding session. Refactoring is simplified due to extensive use of automated test cases.
g. *Continuous Integration:* New features and changes are worked into the system immediately. XP teams focus on validation of software at all times. Programmers develop softwares by writing tests first, and then code that fulfills the requirements reflected in the tests. And customers provide acceptance tests that enable them to be certain that features they need are provided.
h. *Collective Code Ownership:* It idea that all developers own all of the code and enables refactoring.
i. *Pair Programming:* XP Programmers write all production in pairs, two programmers working together at one machine.
j. *Coding Standards:* All code should look the same, it should not possible to determine who coded what based on the code itself
k. *On site Customer:* It acts to "steer" the project as customer provides quick and continuous feedback to the development team.
l. *40 Hour Week:* The work week should be limited to 40 hours. Regular overtime is a symptom of a problem and not a long term solution.

D. 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 act as local admin.

XP Programmer: Writes tests and code.

XP Tracker: Tracks iterations such as program/ project manager, and provides gives feedback on accuracy of estimates and traces progress of iterations.

XP Tester: Helps customer write functional tests and runs functional test and maintains testing tools.

XP Consultant: An external member who guides the team to solve problems. Manager makes decisions.

E. 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 very important practice in XP. However, it cannot be applied to one-developer-projects. Customer collaboration is not very strong. Testing and code development is done by the same person. All the possible problems may not be found because the developer tests from the same perception the product is built.

2.2 SCRUM

A. Background

It was first described by Jeff Sutherland, Ken Schwaber, Mike Beedle in 1996 [7]. It focuses on Agile project management technique rather than development aspect of projects. Scrum is more focused on managerial skills of both managers and developers. Thus, theoretically, scrum can be applied to any industry.

B. Philosophy and Scope

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 light weight software development process consisting of implementing a small number of customer requirements in 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 [8].

Teams should be small comprising about seven to ten people. It can be scaled to larger numbers. The methodology is aimed at project management in changing environments.

C. 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: Scrum strongly supports the values of Respect, Commitment, Focus, Courage, and Openness.
Techniques: Team creation, Backlog creation, Project segmentation, Scrum meetings, Burn down charts.
4 Meetings: Sprint planning meeting, daily scrum meeting, sprint review meeting and sprint retrospective meeting.

Scrum Artifacts:

a. *Product backlog:* It is the complete list of requirements – including bugs, enhancements requests, and usability and performance improvements – that are not currently in the product release.
b. *Sprint backlog:* It is the list of backlog items assigned to a sprint, but not yet completed in common practice; no sprint backlog item should take more than two days to complete. The sprint backlog helps the team predict the level of effort required to complete a sprint.
c. *Burn down chart:* This chart is updated every day, shows the work remaining within the sprint. The burn down chart is used to track sprint progress and to decide when items must be removed from the sprint backlog and deferred to the next sprint.
d. *Release backlog:* Requirements pulled from the product backlog and identified and prioritized for an upcoming release. The release backlog contains more details about the requirement and low level estimate which are usually estimated by the team performing the work.

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

D. Roles and Responsibilities

Scrum Master: Responsible to facilitate the communication between product owner and team and ensures Scrum practices and values are followed up to the end of 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 participates in setting of goals and requirements.

E. 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.
- Scrum does not explicitly address the issue of criticality.
- Customer is offsite and tight customer collaboration is not possible. Also improved team dynamics enabled by Scrum are not available in one-developer project.

2.3 LEAN SOFTWARE DEVELOPMENT (LSD)

A. Background

It was adapted from Lean manufacturing of TOYOTA production system and Bob Charette's Lean development in the 1980s. Lean Software Development Poppendieck & Poppendieck in 2003 [9]. It focuses on the project management aspects of a project and specifies no technical practices; it integrates well with other agile methodologies, such as XP, that focus more on the technical aspects of software development.

B. Philosophy and Scope

Lean Software Development (LSD) is an iterative methodology that focuses on reducing waste and optimizing the entire process to achieve the maximum possible gain. Lean has a rich history in manufacturing and has gained popularity in software development in recent years. It integrates well with the concept of six sigma [9].
Any software development project where there is need for radical change. Focused at company CEOs. No team size specifications because LD is more of a software development management philosophy than a methodology.

C. Features and Benefits

Team Size and Iteration Length: Since Lean Software Development is more of a management philosophy than a development process, team size and iteration lengths are not directly addressed.
Principles: Lean Software Development promotes seven Lean principles. The methodology revolves around these principles, and all other aspects of Lean are designed to reinforce them. The seven principles are:

a. *Eliminate Waste:* Eliminate anything that does not add customer value.
b. *Build Quality In:* Validate and re-validate all assumptions throughout the process. If a metric or practice no longer has value, discard it.
c. *Create Knowledge:* Use short iterative cycles to provide quick, constant feedback to ensure the right things are being focused on.
d. *Defer Commitment:* Don't make decisions until enough is known to make the decision—a sound understanding of the problem and the trade-offs of potential solutions is required.
e. *Deliver Fast:* Minimize the time it takes to identify a business problem and deliver a system or feature that addresses it.
f. *Respect People:* Empower the team to succeed.
g. *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 and it integrates well with the concept of six sigma.

D. Roles and Responsibilities

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

E. Limitations

- The project is highly dependent on cohesiveness and the individual commitments of the team members therefore team building is critical factor.
- A missing or inappropriate involvement from appropriate business analyst could result in scope creep.

2.4 KANBAN

A. 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.

B. Philosophy and Scope

Kanban is taken from Japanese term which means 'signboard'. Kanban is a visual process management system that tells what to produce, when to produce it, and how much to produce. It is a method for managing the creation of products with an emphasis on continual delivery while not overburdening the development team.

Like Scrum, Kanban is a process designed to help teams work together more effectively. 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.

C. 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 & job titles, and Encourage acts of leadership at all levels.

D. Roles and Responsibilities

Kanban does not prescribe any roles. It depends on company and team to decide. Kanban recommends minimizing the cycle time, so if adding a role helps minimize the cycle time, the role can be added and if it makes the process slower, then the role should not be there. Also, if the cost of the role is higher than the value of improved cycle time, then it's unnecessary overhead.

E. 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.

2.5 ADAPTIVE SOFTWARE DEVELOPMENT (ASD)

A. Background

It was described by Jim Highsmith and Sam Bayer in 2000 [10]. ASD is part of rapid application development and focuses on rapid creation and evolution of software systems.

B. Philosophy and Scope

ASD is a method for creating and developing software systems. It offers solutions for the development of large and complex systems by incremental and iterative development, with constant prototyping. It involves product initiation, adaptive cycle planning, concurrent feature development, quality review, and final quality assurance and release. It relies on team members' work at peer level. Usually customer doesn't have all the requirements in the beginning and act as a member with the concept of progressive elaboration.

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

C. Features and Benefits

Phases: Speculate (initiation and planning), Collaborate (concurrent feature development) and Learn (quality review).

Lifecycle Characteristics: Mission focused, Time boxed, Risk driven, iterative, Change tolerant, Feature based.

D. Roles and Responsibilities

Executive Sponsor: Person with overall responsibility for the product.

Participants: in joint application development sessions

Developer: programmer.

Customer: User

2.6 FEATURE DRIVEN DEVELOPMENT (FDD)

A. Background

It arose in the late 1999 from collaboration between Jeff DeLuca and Peter Coad and was later described by Palmer and Felsing in 2002 [11, 12]. It focuses on the domain model, there are five activities: Develop an overall model, Build a list of features, Plan by feature, Design by feature and Build by feature.

B. Philosophy and Scope

FDD focuses on the design and building phases, emphasizes quality aspects throughout the process and includes frequent and tangible deliveries, along with accurate monitoring of the progress of the project. It is simple to understand but powerful approach to build the product or solutions [13].

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

C. Features and Benefits

Team size: Team size varies depending on the complexity of the feature at hand. DeLuca stresses the importance of premium people, especially modeling experts.

Iteration length: Up to two weeks.
Phases: The five phases are Develop overall model; Build the Feature, Plan by Feature, Design by Feature and Build by Feature.
Practices: Domain Object Modeling, Developing by Feature, Individual Class (Code) Ownership, Feature Teams, Inspections, Configuration Management, Regular Builds, Visibility of progress and results
Activities: Develop Overall Model, Build Feature List, Plan By Feature, Design By Feature, Build By Feature, Milestones.
Values: A system for building systems is necessary in order to scale to larger projects, A simple, well-defined process works best, Process steps should be logical and their worth immediately obvious to each team member, Process pride can keep the real work from happening, Good processes move to the background so team members can focus on results, Short, iterative, feature-driven life cycle are best.

D. Roles and Responsibilities

Project Manager: Responsible for all administrative aspects of the project, including the financial and reporting ones.
Chief Architect: Responsible for the overall design of the system, including running all design sessions, code reviews, and technology decisions.
Development Manager: On the hook for the daily development activities. Coordinating the development team and their activities, and dealing with resource issues.
Chief Programmer: A senior developer involved in on-going design and development activities, and is assigned to be responsible for one or more Feature Sets.
Class Owner: A developer who works under the direction of a Chief Programmer to design, code, test, and document features as they are implemented.
Domain Expert: Any business related stakeholder who can define required features that the system should provide. These would include clients, users, and various analysts; anyone who has knowledge of how the business works that could impact the system.
Tester: Responsible for verifying that each feature performs as defined.
Deploy Manager: Deals with not only the actual deployment of code to various environments, but also the definition and/or conversion of data from one format to another.
Technical Writer: Responsible for creating and maintaining all the online and printed documentation that a user will need for the final system.

E. Limitations

- The FDD prescription does not specifically address project criticality.

2.7 DYNAMIC SYSTEMS DEVELOPMENT METHOD (DSDM)

A. Background

It was described by Stapleton in 1995 [14]. DSDM was initially created in 1994 through collaboration of a large number of project practitioners who were seeking to build quality into Rapid Application Development (RAD). It focuses upon early delivery of real benefits to the business.

B. Philosophy and Scope

DSDM is an iterative and incremental methodology that combines the project and product management life cycle into one best process. It was developed to fill in some of the gaps in the RAD method by providing a framework which takes into account the entire development cycle. It is a proven framework for agile project management and delivery, helping to deliver results quickly and effectively as it concentrates on strategic goals and incremental delivery of real business benefits while keeping control of time, cost, risk and quality.
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 the use of DSDM. DSDM is especially suited to projects with changing requirements. It is critical that these requirements be capable of being prioritized.

C. 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
Phases: The DSDM lifecycle has six stages: Pre-project, Feasibility Study, Business Study, Functional Model Iteration, Design and Build Iteration, Implementation, and Post-project.
Principles: DSDM relies on nine principles: active user involvement, teams must be empowered to make their own decisions, frequent delivery of releases is more important than maximizing quality, the primary criteria for deliverables is meeting the business needs, iterative development is essential to reach a correct solution, any change during development can be reversed, the most high level requirements should be unchangeable, testing shall occur throughout the lifecycle of the project, all stakeholders must cooperate and communicate.
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.
This is heavier than XP and Scrum. It provides a technique-independent process and is flexible in terms of requirement evolution. It is efficient in terms of budget and time.

D. Roles and Responsibilities

Executive Sponsor: An important role from the user organisation who has the ability and responsibility to commit appropriate funds and resources. This role has an ultimate power to make decisions.
Visionary: The one who has the responsibility to initialise the project by ensuring that essential requirements are found early on. Visionary has the most accurate perception of the business objectives of the system and the project. Another task is to supervise and keep the development process in the right track.
Ambassador: User Brings the knowledge of user community into the project, ensures that the developers

receive enough amount of user's feedbacks during the development process.
Advisor: User can be any user that represents an important viewpoint and brings the daily knowledge of the project.
Project Manager: Can be anyone from user community or IT staffs who manages the project in general.
Technical Co-ordinator: Responsible in designing the system architecture and control the technical quality in the project.
Team Leader: Leads his team and ensures that the team works effectively as a whole.
Solution Developer: Interpret the system requirements and model it including developing the deliverable codes and build the prototypes.
Solution Tester: Checks the correctness in a technical extent by performing some testing's, raise defects where necessary and retest once fixed. Tester will have to give some comments and documentation.
Scribe: Responsible to gather and record the requirements, agreements, and decisions made in every workshop.
Facilitator: Responsible in managing the workshops progress, acts as a motor for preparation and communication.
Specialist Roles: Business Architect, Quality Manager, System Integrator, etc.

E. Limitations

- It is based on user involvement which is not possible in every project
- Because of its strictness and eight principles, the main problem with DSDM is that it can be restrictive and difficult to work with compared to other agile development software methods.

2.8 AGILE MODELING (AM)

A. Background

Agile Modeling (AM) is proposed by Scott Ambler in 2002 [15]. It focuses only on documentation and modeling. It can be used with any software development process as it's not a complete software development method.

B. Philosophy and Scope

AM is a new approach for performing modeling activities. It attempts to adapt modeling practices using an agile philosophy. It is a practice-based methodology for effective modeling and documentation of software-based systems. At a high level, it is a collection of best practices and at a more detailed level, it is a collection of values, principles, and practices for modeling software that can be applied on a software development project in an effective and light-weight manner. It is a supplement to other agile methodologies such as Scrum, XP, and RUP and explicitly included as part of the Disciplined Agile Delivery (DAD) framework [16, 17]. The aim is to keep the amount of models and documentation as low as possible. Cultural issues are addressed by depicting ways to encourage communication, and to organize team structures.
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.

C. Features and Benefits

Team Size and Iteration Length: Since AM is not a complete software process development method and should be used with other development methods, the team size, exact iteration lengths, distribution and system criticality will depend on the development process being used

Values: include those of XP - communication, simplicity, feedback and courage; and also include humility.

Goals: The three main goals of AM are to define and show how to put into practice a collection of values, principles and practices that lead to effective and lightweight modelling; to address the issue on how to apply modeling techniques on Agile software development processes; and to address how you can apply effective modeling techniques independently of the software process in use.

D. 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.

E. Limitations

Agile Modeling disciplines can be difficult to apply:

- On large teams (> 30) without adequate tooling support.
- Where team members are unable to share and collaborate on models (which would make Agile Software Development in general difficult).
- When modeling skills are weak or lacking.

2.9 CRYSTAL

A. Background

It was developed by Alistair Cockburn in 2001. It focuses more on people rather than process.

B. Philosophy and Scope

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 that focuses on communication in small teams developing software that is not life-critical. Crystal methods involve frequent delivery; reflective improvement; close communication; personal safety; focus; easy access to expert users; and a technical environment with automated testing, configuration management, and frequent integration [17].
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.

C. Features and Benefits

Team size: The Crystal Family accommodates any team size; however, Cockburn puts a premium on premium people. 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: Up to 4 months for large, highly critical projects.

Properties: Frequent delivery, Reflection improvement, Osmotic communication, Personal safety, Focus, Easy access to expert users, 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

D. Roles and Responsibilities

Sponsor: Finances the project and delivers the mission statement.
Senior Designer: Maintains team structure, implements methodology, designs the system.
Designer/Programmer: Creates screen drafts, design sketches and notes, common object model, source code, packaged system, migration code, and test cases.
User: Helps 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.

E. Limitations

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

2.10 AGILE UNIFIED PROCESS (AUP)

A. Background

Agile Unified Process (AUP) is a simplified version of the IBM Rational Unified Process (RUP) developed by Scott Ambler. Introduced in 2005 and revised in 2006 [18]. The major enhancements focus is on real-time and web-based development.

B. Philosophy and Scope

AUP describes a simple, easy to understand approach to developing business application software using agile techniques and concepts yet still remaining true to the RUP. In 2012, the AUP was superseded by Disciplined Agile Delivery. Since then work has ceased on evolving AUP.

The AUP applies agile techniques including test driven development (TDD), Agile Modeling, agile change management, and database refactoring to improve productivity.

C. Features and Benefits

Disciplines: Each iteration consists of seven work areas or disciplines performed during the iteration. Disciplines are performed in an iterative manner, defining the activities which development team members perform to build, validate, and deliver working software which meets the needs of their stakeholders. The seven disciplines are:

a. *Model.* The goal is to understand the business of the organization, the problem domain being addressed by the project, and identify a viable solution to address the problem domain.
b. *Implementation.* The goal is to transform model(s) into executable code and perform a basic level of testing, in particular unit testing.
c. *Test.* The goal is to perform an objective evaluation to ensure quality. This includes finding defects, validating that the system works as designed, and verifying that the requirements are met.
d. *Deployment.* The goal is to plan for the delivery of the system and to execute the plan to make the system available to end users.
e. *Configuration Management.* The goal is to manage access to project artifacts. This includes not only tracking artifacts versions over time but also controlling and managing changes to them.
f. *Project Management.* The goal 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 on time and within budget.
g. *Environment.* The goal is to support the rest of the effort by ensuring that the proper process, guidance (standards and guidelines), and tools (hardware, software, etc.) are available for the team as needed.

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

AUP Philosophies

a. *Your staff knows what they're doing*. People are not going to read detailed process documentation, but they will want some high-level guidance and/or training from time to time. The AUP product provides links to many of the details, if you are interested, but doesn't force them upon you.
b. *Simplicity.* Everything is described concisely using a handful of pages, not thousands of them.
c. *Agility.* The Agile UP conforms to the values and principles of the agile software development and the Agile Alliance.

d. Focus on high-value activities. The focus is on the activities which actually count not every possible thing that could happen to you on a project.

e. Tool independence. You can use any toolset that you want with the Agile UP. The recommendation is that you use the tools which are best suited for the job, which are often simple tools.

f. You'll want to tailor the AUP to meet your own needs.

AUP Phases

The overall development cycle consists of four phases:

Phase 1: Inception: The goal is to identify the initial scope of the project, a potential architecture for your system, and to obtain initial project funding and stakeholder acceptance.

Phase 2: Elaboration: The goal is to prove the architecture of the system.

Phase 3: Construction: The goal is to build working software on a regular, incremental basis which meets the highest-priority needs of your project stakeholders.

Phase 4: Transition: The goal is to validate and deploy your system into your production environment.

Each phase can be further broken down into Iterations. Iteration is a complete development loop resulting in a release of an executable increment to the system.

AUP Releases

The Agile Unified Process distinguishes between two types of iterations: A development release iteration results in a deployment to the quality-assurance and/or demo area and a production release iteration results in a deployment to the production area. This is a significant refinement to the Rational Unified Process.

D. Limitations

- Tackling model inconsistency is not explicitly addressed.
- Modeling may risk agility if limits are not strictly observed; the list of AUP's models that are produced as the absolute minimum is actually quite long.
- The AUP isn't for everyone. XP users will likely find the AUP fairly heavy, and "traditional RUP" users may find that it's too streamlined. If you want something lighter, then XP is highly suggested. If detailed, well-defined software process is required, then it is highly suggested to consider RUP. Management seems to prefer RUP than developers due to the large number of artifacts and has a lot to offer, and can be cut down to something quite useful. The AUP lands between the two, adopting many of the agile techniques of XP and other agile processes yet retaining some of the formality of the RUP.

III. ADVANTAGES OF AGILE SOFTWRE DEVELOPMENT

The key benefits of adopting agile methodology [24, 25] in software development processes explained in detail as follows:

1. **Rapid, iterative and incremental delivery:** Project delivery is divided into small functional releases to check functionality, to manage risk and to get early feedback from customer and end users. These new and small features releases are delivered quickly and frequently on a fixed schedule iterations of 1-4 weeks each, with a high level of predictability. Project plans, requirements, design, code and tests are created initially and updated incrementally as required to adapt to project changes. This helps in checking and monitoring the software functionality progress frequently rather than at end of long milestones.
2. **Increased performance:** Daily stand-up meetings provide an opportunity to exchange valuable information and to fine tune improvements continuously. The ability to discuss complex projects through simple stories and simple design encourages teamwork. Frequent and better communication leads to increased knowledge sharing and trust among team members which increases the team productivity and generates better performance in terms of good Return on Investment.
3. **Flexibility of design:** Flexibility is based on the development process used for the project 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 that can handle changes easily.
4. **Adaptive to the changing environment:** Using agile software development approach, software is developed over several iterations. Each iteration is characterized by analysis, design, coding and testing. The working software is delivered to the customer and end user for their use and feedback after every iteration. Agile approach encourages and implements any change requirement from the customer at any stage of development to upgrade the software.
5. **Reduces risks of development:** As the incremented mini software is delivered to the customers after every short development cycle and feedbacks are taken from the customers, it warns developers about the upcoming problems which may occur at the later stages of development. It also helps to discover errors quickly and they are fixed immediately.
6. **Working software:** Agile development's commitment to the delivery of working, tested software at recurrent intervals ensures a much greater reliability and opportunity to incorporate the user and technology-driven feedback. Agile practices focus on working software that provides greater feedback which makes agile projects easier to manage and customer gets the best product as learning is incorporated.
7. **Ensures customer satisfaction:** Agile methodology encourages active customer involvement throughout the software development lifecycle. The deliverables developed after each iteration is given to the user for use and improvement is done based on the customer feedback only. So at the end what we get as the final product is of high quality and ensures the customer satisfaction as the entire software is developed based on the requirements taken from customer.

8. **Avoids over production:** The traditional system requirement document is still built where many features are not wanted or required. These "low" or "no" value features are at the bottom of the backlog but they still get built in Waterfall. On contrary, Agile approach builds the best product by building it for now and not later.
9. **Improvement in quality:** Breaking down the project into manageable units or sprits allows the project team to focus on high quality development, testing, and collaboration. Quality is also improved by producing frequent builds and conducting continuous testing and customer feedback during each iteration. Test-driven development and refactoring is often used in finding and fixing defects quickly and identifying expectation mismatches early that leads to higher code reuse and better quality.
10. **Least documentation:** The documentation in agile methodology is short and to the point as internal design of the software is usually not documented. The main content in the documentation are product features list, duration for each iteration and date which saves the development time and deliver the project in least possible time.
11. **Fault detection:** Organizational processes demand high quality bug free software. A continuous testing and integration characteristic of agile methodologies such as XP enforces the delivery of high quality bug free software. As testing is performed during each iteration, error and faults are identified earlier and are fixed instantaneously before it increases in severity. Automated tests find regression defects earlier, continuous integration finds defects earlier, pair Programming finds defects much sooner earlier. Also, continuous testing allows continuous testing feedback, which further improves code developed in future iterations.
12. **Best practices:** Incorporating some well-known Agile practices can help the teams employ highly competent, well-tested applications across the required spectrum of platforms and devices. Agile forces "architecture killers" to the start of the project. Better to fail early, not late - when all the money has been spent, most changes have low cost of change.

Agile software development approach not only provides benefits to the development team, but also provides number of business benefits to the client. Agile addresses many of the most common project drawbacks, such as budget control, schedule predictability and scope creep.

IV. DISADVANTAGES OF AGILE SOFTWARE DEVELOPMENT

Besides many advantages of using agile software development method and like traditional methods, the agile methods also have some limitations and have also been criticized by some practitioners and researchers, mainly focusing on following aspects [6, 10, 11, 20, 21, 23, 24, 25]:

1. **Not suitable for large projects:** An agile development method is suitable for small teams, but does not scale well to larger projects, as numerous iterations are needed to complete the desired functionality. On a large scale project, opportunity cost to employ agile methods may be too high for a foregone production on more profitable and lean projects.
2. **Customer interaction:** Agile requires active user involvement and close collaboration with the project team throughout the software development cycle. This practice of Agile is very rewarding and ensures delivery of the right product. However, in practice, these principles are very demanding on the user representative's time and require a big commitment for the duration of the project.
3. **Insufficient and unclear requirements:** Agile requirements are usually insufficient and unclear which eliminates wasted effort on deliverables that don't last which controls budget and schedule. The requirements are clarified and specified during the development phase just in time and documented in much less detail due to the timeliness of conversations. However, this provides limited information to new starters in the team about product features and how they should work. Agile methodology is based on customer involvement because the entire project is developed according to the requirements given by the customers. So if the customer representative is not clear about the product features and final outcome, the development process and project can easily get taken off track.
4. **Changing requirements:** Requirements emerge and evolve throughout the development lifecycle which means flexibility to makes change as needed to ensure 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 the maturity of a strong and clear vision, and the discipline of fixing schedule, budget and scope is harder. Agile is time consuming and can cause wastage of resources because of constant change of requirements. If the customers are not satisfied by the partial software developed by certain iteration and they change their requirements then that incremented part is of no use. So it is the total wastage of time, effort and resources required to develop that increment.
5. **Difficulty in integration testing:** Testing is integrated throughout the product development lifecycle which ensures the quality throughout the project without the need for an extensive test phase at the end of the project. This practice of Agile demands testers throughout the project which increases the cost of resources on the project. The cost of a long and unpredictable test phase can cause huge unexpected costs when a project over-runs. It does have the effect of reducing some very significant risks that have proven through research to cause many projects to fail. Also, there is an additional cost to the project to adopt continuous testing throughout.
6. **Frequent delivery:** Adopting an Agile method promotes frequent delivery of product, followed by User Acceptance Testing (UAT) for which the product

owner needs to be available for prompt testing of the features as they are delivered and throughout the entire duration of the project. This exercise can be quite time consuming but helps drastically to ensure a quality product that meets user expectations. The feedback is that agile practice is rather intense for developers as they are required to complete each feature 100% within each iteration, and the endlessness of iterations can be mentally tiring, so it's important to find a sustainable pace for the team.

7. **Lack of documentation:** Though the least documentation saves lot of development time as an advantage of Agile method but it is a big disadvantage for the developer. Since internal design changes frequently based upon users changing requirements after every iteration, it is not possible to maintain the detail documentation of design and implementation because of project deadline. Therefore, due to limited available information, it is very difficult for the new developers who join the development team at the later stage to understand the actual method followed to develop the software.
8. **More helpful for management than developer:** The agile methodology helps management to take decisions about the software development, set goals for developers and fix the deadline for them. But it is very difficult for the baseline developers to cope up with the ever changing environment, changing design and code based on just in time requirements. Management overhead is also increases as successful application of an agile methodology requires strong teamwork and the project manager should always be involved in the dynamics of the team.
9. **Culture and co-located teams:** Many times application development teams are located in different parts of the globe, making in person face to face communication almost impossible. To overcome this problem, most developers use video conferencing tool to deliver high quality product.
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. Another limitation is that agile methodologies concentrate work quality on the skills and behaviours of the developers, as the design of the modules and sub-modules are created mainly by single developer. Agile methodologies will not provide the best solution, when developing reusable software as it focuses in building a system to solve specific problems, and not the general ones. So, Agile works best for relatively small team's members as compared to large teams.
11. **Traditional waterfall development mind set:** Organizations steeped in waterfall development mind set are reluctant to adopt agile methodologies. In Agile software development method, the main emphasis is on development rather than design. It basically focuses on processes for getting requirements and 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 advantages of applying agile methodologies in the development, there is a set of assumptions that are assumed to be true. To mention some are: cooperation and face to face relation between the customers and the development team; evolving and changing requirements of the project; developers having good individual skills and experiences. If these assumptions do not apply to a software development project, then it is better to look for other methodologies to apply for the development process, in order to get better results.

V. CONCLUSION

Traditional software development approaches have a potential to provide straightforward, systematic, and organized process in the software development. However, the traditional approaches have limitations, which include slow adaptation to rapidly changing business requirements, a tendency to be over budget and behind schedule, a lack of dramatic improvements in productivity, reliability, and simplicity.

Agile development methodology is a conceptual framework for undertaking any software engineering project. In general agile approach can provide a shorter development cycle, higher customer satisfaction, and quicker adaptation to rapidly changing business requirements. It also attempts to minimize risk and maximize productivity by delivering working software in short iterations that increases the internal and external practices of the software development.

A selection and implementation of appropriate process is crucial as it ensures the organization to maximize their chance to deliver their software successfully with long term implications.

REFERENCES

[1] Agile Alliance. "Principles behind Agile Manifesto". www.agilemanifesto.org/principles.html, 2006.
[2] Beck, Kent, "Manifesto for Agile Software Development". Agile Alliance, http://agilemanifesto.org, 2001.
[3] Beck, K. and C. Andres, "Extreme Programming Explained" (2nd Edition), Addison-Wesley Professional, 2004.
[4] Beck, Kent, "Extreme Programming Explained: Embrace Change", Addison-Wesley, ISBN 0-201-61641-6, 2000.
[5] Beck, Kent, "Extreme Programming Explained: Embrace Change", second ed., Addison-Wesley, ISBN 978-0321278654, 2004.
[6] D. Cohen, M. Lindvall, P. Costa, "An introduction to agile methods, in: M.V. Zelkowitz (Ed.)", Advances in Computers, Advances in Software Engineering, vol. 62, Elsevier, Amsterdam, 2004.
[7] Schwaber, K. and M. Beedle, "Agile Software Development with Scrum", Prentice Hall PTR, 2001.
[8] K. Schwaber, M. Beedle, "Agile Software Development with Scrum", Prentice Hall, Upper Saddle River, 2001.
[9] M. Poppendieck, T. Poppendieck, "Lean Software Development: An Agile Toolkit for Software Development Managers", Addison-Wesley, Boston, ISBN 0-321-15078-3, 2003.
[10] Highsmith, J., "Adaptive software development: a collaborative approach to managing complex systems", Dorset House Publishing Co., Inc, 2000.
[11] Coad, P., J. deLuca, et al., "Java Modeling Color with Uml: Enterprise Components and Process with CDROM", Prentice Hall PTR, 1999.
[12] Palmer, S. R. and M. Felsing, "A Practical Guide to Feature-Driven Development", Pearson Education, 2001.

[13] S.R. Palmer, J.M. Felsing, “A Practical Guide to Feature-driven Development”, Prentice Hall, Upper Saddle River, NJ, ISBN 0- 13-067615-2, 2002.
[14] A. P. Framework and I. DSDM, “Introduction Introducing the DSDM ® Agile Project Framework”.
[15] Agile Modeling (AM) Home Page, “Effective Practices for Modeling and Documentation” (http://www.agilemodeling.com/).
[16] Ambysoft. “The Agile System Development Life Cycle (SDLC)”, http://www.ambysoft.com/essays/agileLifecycle.html#Cycle0, 2011.
[17] A. Cockburn, “Crystal Clear: A Human-Powered Methodology for Small Teams”, Addison-Wesley, ISBN 0-201-69947-8, 2004.
[18] Waters, John K, "Agile lands role in games and business software", 2008.
[19] http://www.ambysoft.com/unifiedprocess/agileUP.html.
[20] H. Merisalo-Rantanen, T. Tuure, R. Matti, “Is extreme programming just old wine in new bottles: a comparison of two cases, Journal of Database Management” 16 (4) (2005) 41–61, 2005.
[21] M. Stephens, D. Rosenberg, “Extreme Programming Refactored: The Case Against XP”, Apress, Berkeley, CA, ISBN 1-59059-096-1, 2003.
[22] P. Mcbreen, “Questioning Extreme Programming”, Pearson Education, Boston, MA, USA, ISBN 0-201-84457-5, 2003.
[23] Kelly Waters, “Disadvantages of Agile Development”, http://www.allaboutagile.com/disadvantages-of-agile-development. Sept. 2007.
[24] Gaurav Kumar, Pradeep Kumar Bhatia, “Impact of Agile Methodology on Software Development Process”, International Journal of Computer Technology and Electronics Engineering (IJCTEE) Volume 2, Issue 4, August 2012.
[25] Sheetal Sharma”Agile Processes and Methodologies: A Conceptual Study”, International Journal on Computer Science and Engineering (IJCSE). Vol. 4 No. 5. May 2012.

.