International Journal of Engineering and Technical Research (IJETR)
ISSN: 2321-0869, Volume-3, Issue-3, March 2015

AGILE: Software development model

Abhilasha yadav

***Abstract*— We all know that in recent modern era software is the basic requirement for operating almost every digital machine. So software development life cycle is also required. The purpose of this paper is to explain SDLCs and their types. It also list down merits and demerits of SDLCs. And it also explains AGILE Software development life cycle. How it is better than other SDLCs which type of merits it have? How it helps to overcome the loopholes present in other SDLCs? This research basically focused on AGILE. Researcher want to explain almost every point related to the AGILE software development life cycle.**

***Index Terms*— Software Development Life Cycle, SDLC models, Agile Manifesto, Different Styles of Agile.**

I. INTRODUCTION

Process of Software development starts when someone feels the requirement of software and end when the use of that software finished. For making good software there are many steps which have to follow. These steps are:

1). Planning 2). Requirement 3). Design phase 4). Coding 5). Testing 6). Implementation / Maintenance. These phases are base of a software development.

There are many software development life cycles presented till now. These SDLCs are applicable on different-different situations for software development. These SDLCs are:

- Waterfall SDLC
- Prototype SDLC
- Iterative SDLC/ Incremental SDLC
- Spiral SDLC
- V-SDLC

All above SDLCs are good to produce software and in efficient manner but these are traditional SDLCs and problem with these are time consumption and large documentation. So overcome these problems Agile SDLC is proposed. In this paper we discussed about agile and its type. How agile is better than other? Section 1 is introduction; section 2 is short description of SDLCs and loopholes of these SDLCs; in section 3 we discussed Problem associated with traditional methods in section 4 we describe Agile SDLC and its type; in section 5 we describe that which loopholes covered by Agile and how it help to overcome; in section 6 we discuss about future work in Agile; section 7 is conclusion.

Manuscript received March 02, 2015.

Abhilasha yadav,Sri Satya Sai Institute of Science & Technology, Sehore.

II. SDLC

SDLC, Software Development Life Cycle is a process used by software industry to design, develop and test high quality software. The SDLC aims to produce high quality software that meets or exceeds customer expectations, reaches completion within times and cost estimates.

A. *Waterfall SDLC*

The Waterfall Model was first Process Model to be introduced. It is also referred to as a linear-sequential life cycle model. It is very simple to understand and use. In a waterfall model, each phase must be completed before the next phase can begin and there is no overlapping in the phases. [18, 22]

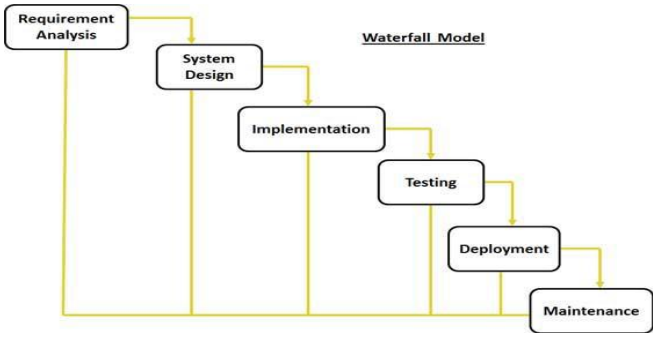

Fig1 waterfall model

1) *Requirement analysis:* In this phase all possible requirements of the system are being collected in this phase and documented in a requirement specification doc.
2) *System Design***:** In this phase blueprint of system is created. This phase helps in specifying hardware and system requirements and also helps in defining overall system architecture.

3) *Implementation*: In this phase the system is first developed in small programs called units, each unit is developed and tested for its functionality which is referred to as Unit Testing.

4) *Integration and Testing***:** after implementation of all units these units are integrated into a single system. And after that the entire system is tested for any faults and failures.
5) *Deployment of system***:** Once the functional and non functional testing is done, the product is deployed in the customer environment or released into the market.
6) *Maintenance***:** There are some issues which come up in the client environment. To fix those issues patches are released. Also to enhance the product some better versions are released. Maintenance is done to deliver these changes in the customer environment.

B. *Iterative SDLC/ Incremental SDLC***:**

An iterative life cycle model does not attempt to start with a full specification of requirements. Instead, development begins by specifying and implementing just part of the software, which is then reviewed in order to identify further requirements. This process is then repeated, producing a new version of the software at the end of each iteration. [18, 22]

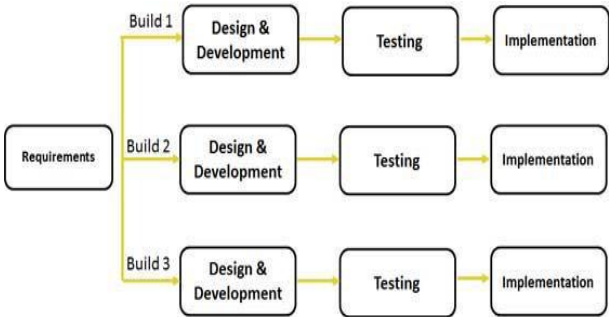

Fig2 Iterative SDLC/Incremental SDLC

C. *Spiral SDLC*

Spiral model is a combination of iterative development process model and sequential linear development model i.e. waterfall model with very high emphasis on risk analysis. It allows for incremental releases of the product, or incremental refinement through each iteration around the spiral.

The spiral model has four phases these phases are: [18, 22]

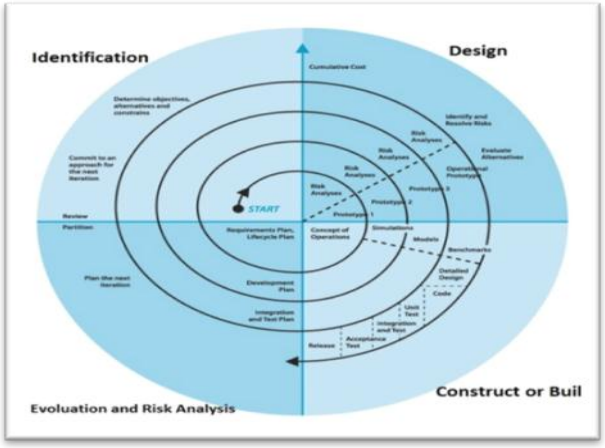

Fig3 Spiral-SDLC

1) *Identification*: This phase starts with gathering the business requirements in the baseline spiral. This also includes understanding the system requirements by continuous communication between the customer and the system analyst.
2) *Design*: Design phase starts with the conceptual design in the baseline spiral and involves architectural design, logical design of modules, physical product design and final design in the subsequent spirals.
3) *Construct or Build*: Construct phase refers to production of the actual software product at every spiral. In the baseline spiral when the product is just thought of and the design is being developed a POC (Proof of Concept) is developed in this phase to get customer feedback.
4) *Evaluation and Risk Analysis*: Risk Analysis includes identifying, estimating, and monitoring technical feasibility and management risks, such as schedule slippage and cost overrun. After testing the build, at the end of first iteration, the customer evaluates the software and provides feedback.

D. *V-SDLC*

V -Model is an extension of the waterfall model and is based on association of a testing phase for each corresponding development stage. This means that for every single phase in the development cycle there is a directly associated testing phase. This is a highly disciplined model and next phase starts only after completion of the previous phase.

Under V-Model, the corresponding testing phase of the development phase is planned in parallel. So there are Verification phases on one side of the 'V' and Validation phases on the other side. Coding phase joins the two sides of the V-Model.

Following are the Verification phases in V-Model:[18, 22]

1) *Business Requirement Analysis*: This is the first phase in the development cycle where the product requirements are understood from the customer perspective.
2) *System Design*: After getting clear and detail about product requirements, then it's time to design the complete system. System design would comprise of understanding and detailing the complete hardware and communication setup for the product under development.
3) *Architectural Design*: Architectural specifications are understood and designed in this phase. This is also referred to as High Level Design (HLD).
4) *Module Design*: In this phase the detailed internal design for all the system modules is specified, referred to as Low Level Design (LLD).

5) *Coding Phase:* In the Coding phase the best suitable programming language is decided based on the system and architectural requirements. The coding is performed based on the coding guidelines and standards.
6) *Validation Phases*: Following are the Validation phases in V-Model:

- *Unit Testing*: Unit testing is the testing at code level and helps eliminate bugs at an early stage, though all defects cannot be uncovered by unit testing.
- *Integration Testing*: Integration tests are performed to test the coexistence and communication of the internal modules within the system.

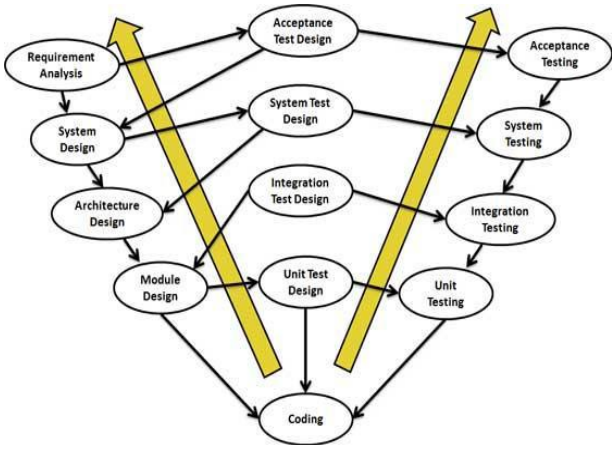

Fig4 V-SDLC

- *System Testing*: System tests check the entire system functionality and the communication of the system under development with external systems. Most of the software and hardware compatibility issues can be uncovered during system test execution.
- *Acceptance Testing*: Acceptance tests uncover the compatibility issues with the other systems available in the user environment.

III. PROBLEM ASSOCIATED WITH TRADITIONAL METHODS [18, 22]

1) *Waterfall Model*: The major weaknesses of the Waterfall Model are

- No working software is produced until late during the life cycle.
- High amount of risk and uncertainty.
- Not a good model for complex and object oriented subject.
- Poor model for long and ongoing projects.
- Not suitable for the project where requirements are at a moderate to the high risk of changing. So risk and uncertainty is high with this process model.
- It is difficult to measure progress within stage.
- Cannot accommodate changing requirements.
- Adjusting scope during the life cycle.
- Integration is done as a "big-bang" at the very end, which does not allow identifying any technology or business bottleneck or challenges early.

2)

erative Model: Loopholes of iterative model are

- Cost of change is lesser but it is not very suitable for changing requirements.
- More management attention is required.
- System architecture or design issues may arise because not all requirements are gathered in the beginning of the entire life cycle.
- Defining increments may require definition of the complete system.
- Not suitable for smaller projects.
- Management complexity is more.
- End of project may not be known which a risk is.
- Highly skilled resources are required for risk analysis.
- Project's progress is highly dependent upon the risk analysis phase.

3) *Spiral*: Loopholes associated with spiral are

- Management is more complex.
- End of project may not be known early.
- Not suitable for small or low risk projects and could be expensive for small projects.
- Process is complex
- Spiral may go indefinitely.
- Large number of intermediate stages requires excessive documentation.

4) *V –Model*: Problems with V-model are

- High risk and uncertainty.
- Not a good model for complex and object-oriented projects.
- Poor model for long and ongoing projects.
- Not suitable for the projects where requirements are at a moderate to high risk of changing
- Once an application is in the testing stage, it is difficult to go back and change a functionality
- No working software is produced until late during the life cycle.

IV. AGILE SOFTWARE DEVELOPMENT LIFE CYCLE

In software application development, agile software development (ASD) is a methodology for the creative process that anticipates the need for flexibility and applies a level of pragmatism into the delivery of the finished product. Agile software development focuses on keeping code simple, testing often, and delivering functional bits of the application as soon as they're ready. The goal of ASD is to build upon small client-approved parts as the project progresses, as opposed to delivering one large application at the end of the project.

Agile development model is also a type of Incremental model. Software is developed in incremental, rapid cycles. This results in small incremental releases with each release building on previous functionality. Each release is thoroughly tested to ensure software quality is maintained. It is used for time critical applications. Extreme Programming (XP) is currently one of the most well known agile development life cycle model.

t

Agile Methodology is a type of project management process. The agile method anticipates change and allows for much more flexibility than traditional methods. Clients can make small objective changes without huge amendments to the budget or schedule. The process involves breaking down each project into prioritized requirements, and delivering each individually within an iterative cycle. An iteration is the routine of developing small sections of a project at a time. Each iteration is reviewed and assessed by the development team and client.

1) *The Values and Principles of the Agile Alliance*: In February of 2001, seventeen practitioners of several programming methodologies came together at a summit in Utah to discuss the problems of existing methodologies, the ways to overcome those, and the values to support agile or lightweight software development at high level; then they published The Agile Manifesto with the four main values that were agreed on as [10]:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

The Agile Manifesto is based on twelve principles [11]:

1. Customer satisfaction by rapid delivery of useful software
2. Welcome changing requirements, even late in development
3. Working software is delivered frequently (weeks rather than months)
4. Close, daily cooperation between business people and developers
5. Projects are built around motivated individuals, who should be trusted
6. Face-to-face conversation is the best form of communication (co-location)
7. Working software is the principal measure of progress
8. Sustainable development, able to maintain a constant pace
9. Continuous attention to technical excellence and good design
10. Simplicity—the art of maximizing the amount of work not done—is essential
11. Self-organizing teams
12. Regular adaptation to changing circumstances

2) Different Styles of Agile Software Development

There are several different approaches of Agile Software Development that focus on different aspects of the projects. In this chapter, four of the most common ones of these styles, namely eXtreme Programming, SCRUM, Adaptive Software Development, Crystal, are presented.

i. eXtreme Programming (XP)

Extreme programming was originally started to be formulated in 1996 by Kent Beck. And Ron Jeffries, describes the method as "Extreme Programming is a discipline of software development with values of simplicity, communication, feedback and courage. We focus on the roles of customer, manager, and programmer and accord key rights and responsibilities to those in those roles". [3, 5]

In contrast to the traditional methods, XP is based on small releases that are produced periodically, while it places much importance on customer satisfaction in parallel with continuous feedback, and thus in XP adopting changes of specifications is significant. Therefore, this naturally implies that the testing to obtain satisfying working releases plays a very crucial role in XP.

These practices help us understand the principles of XP more precisely.

a) *On-site Customer*: XP requires the customers to actively and collaboratively participate to the project at all times. In this way, the team gains the constant availability of the customer that can always quickly provide instructions and answers about the requirements to the development team.[5]

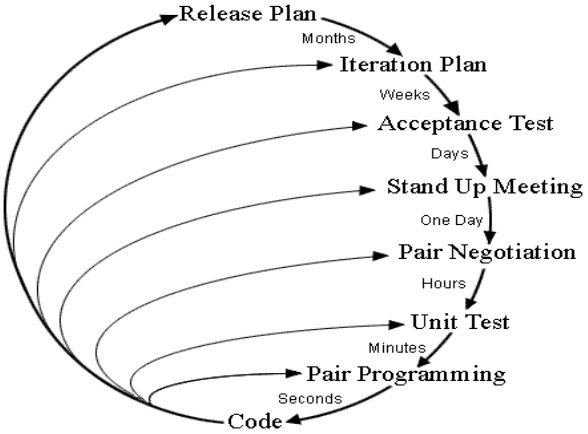

Fig5. Planning and Feedback Loop in eXtreme Programming

b) *Small Releases*: In XP, a project is developed by iteratively putting small but- tested and -working releases that are updated periodically. In this manner, these small releases lead to the early benefit and/or use to the customer, thus to gain early feedback from the customer.[5]

c) *Planning Game:* Inside each release, an Extreme team plans just a few weeks at a time, with clear objectives and solid estimates. XP planning works continuously and iteratively considering the scheduling on the small releases and the goals for the next releases, according to the customer. Thus, this rule leads the customer to steer the development team by choosing the ideal combination of stories within the time and the funds available.[5]

d) *Metaphor:* Metaphor rule aims to define and guide the development with a simple common story to ensure each member of the development team is completely aware of how the entire product works.[4]

e) *Pair programming*: Pair programming means two programmers continuously working on the same code. Pair programming can improve design quality and reduce defects. This shoulder-to-shoulder technique serves as a continual design and code review process, and as a result defect rates are reduced. This action has been widely recognized as continuous code inspection.

f) *Simple Design*: XP developers stress implementing the simplest possible solution always in all stages. Hence, XP avoids the complexity and extra code. Hence, in XP the project is designed in a way as simple as possible. Beck expresses this idea as "Developers are urged to keep design as simple as possible, say everything once and only once.[3]

g) *Collective Code Ownership*: In XP, the term collective ownership of a project means that each part of the code belongs to the whole development team. Thus, needed improvements on other programmers' code can be made by any member of the team while this also leads to a faster progress and cleaner code.[5]

h) *Coding Standard*: In XP, a certain coding standard is used in order to have a common understanding and ability to work on the development. Jeffries supports

this rule as "Coding standard ensures that the code communicates as clearly as possible and supports our shared responsibility for quality everywhere."[5]

i) *Continuous Integration*: Integration of the code is extremely difficult if it is done once at the entire development, due to many lines of code and the difficulty of identifying bugs. Thus, in XP each completed task is integrated to the system right away, then the application is built and tested daily several number of times. In this manner, the system always stays as completely integrated.[5]

j) *Test-Driven Development*: The unit tests are kept in an automated test suite by the programmers; and whenever they change a section of code, the test suite is run to observe immediately whether it caused a problem on what had been working, while the customer evaluates the new parts and give feedback right away.[4]

k) *Refactoring*: Refactoring is the process of improving the structure of the code without changing its function. Thus, refactoring aims to keep the design of the code as simple and understandable as possible, to avoid duplication, and to add flexibility to the code.[5]

ii. SCRUM

Ken Schwaber (1996), the pioneer of SCRUM, states that the development is an unpredictable process, whereas SCRUM produces breakthrough productivity, enabling building the best systems possible in complex, unpredictable environments. Schwaber defines SCRUM as:
"Scrum is a method that aims to help teams to focus on their objectives. It tries to minimize the amount of work people have to spend tackling with less important concerns. Scrum is a response to keep things simple in the highly complicated and intellectually challenging software business environment."[9]
SCRUM consists of short, intensive, daily meetings of the whole project team aiming to deliver as much quality software as possible within a series of short time boxes called "sprints"

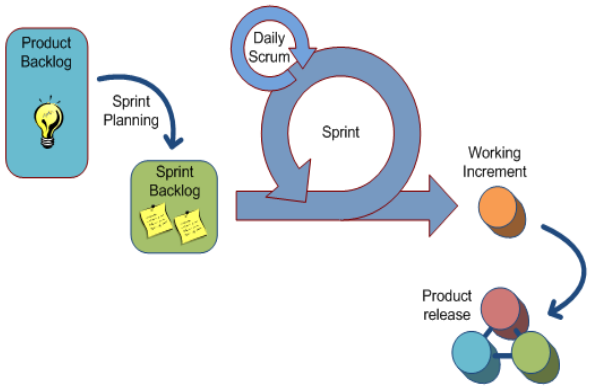

Fig 6 SCRUM Process Stages

Schwaber lists the key principles of SCRUM as follows [9]:

- Small working teams that maximize communication, minimize overhead, and maximize sharing of tacit, informal knowledge
- Adaptability to technical or marketplace (user/customer) changes to ensure the best possible product is produced
- Frequent "builds", or construction of executable, that can be inspected, adjusted, tested, documented, and built on
- Partitioning of work and team assignments into clean, low coupling partitions, or packets
- Constant testing and documentation of a product - as it is built
- Ability to declare a product "done" whenever required (because the competition just shipped, because the company needs the cash, because the user/customer needs the functions, because that was when it was promised...)

iii. Adaptive Software Development (ASD)

In 1992, Jim Highsmith's effort of working on a short interval, iterative, rapid application development process evolved into Adaptive Software Development (ASD)
ASD is an agile method that is based on the continuous change, and is opposed to stable planning, such as Waterfall approach's planning stage. ASD's change-oriented life cycle consists of three main stages as Speculate, Collaborate and Learn. [6, 17]

- *Speculate*
  Speculate stage of ASD's life cycle is used for initiation and cycle planning, as Highsmith explains it with the following seven steps:
  1. Conduct the project initiation phase.
  2. Determine the project time-box.
  3. Determine the optimal number of cycles and the time-box for each.
  4. Write an objective statement for each cycle.
  5. Assign primary components to cycles.
  6. Assign technology and support components to cycles.
  7. Develop a project task list.

- *Collaborate*
  Working software is delivered by concurrent component engineering in ASD where interaction of people and management of interdependencies are crucial for the development as in XP's pair programming and collective code ownership methods.

- *Learn*
  Learning is an iterative step in ASD applied at the end of each development cycle, leading to a loop to adaptive planning for the next cycle and to quality assurance. Highsmith expresses that result quality from both the customer's perspective and a technical perspective are learned in this stage, as well as the functioning of the delivery team and the practices they are utilizing with project's status.

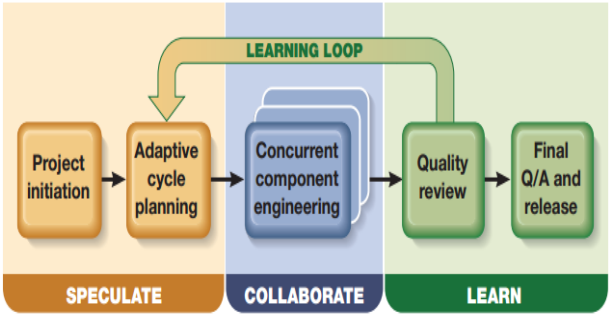

Fig 7 Adaptive Software Development Change Oriented Life Cycle

iv. Crystal

Crystal is a family of human-oriented light-weight methods with efficiency and agility purposes, developed by Alistair Cockburn in early 1990s. Highsmith states that Crystal focuses on collaboration and cooperation using project size, criticality, and objectives to craft appropriately configured practices for each member of the Crystal family of methodologies. [6, 17]

The design principles of Crystal can be summarized as:

The team can reduce intermediate work products as it produces running code more frequently, as it uses richer communication channels between people.

Every project is slightly different and evolves over time, so the methodology, the set of conventions the team adopts, must be tuned and evolve.

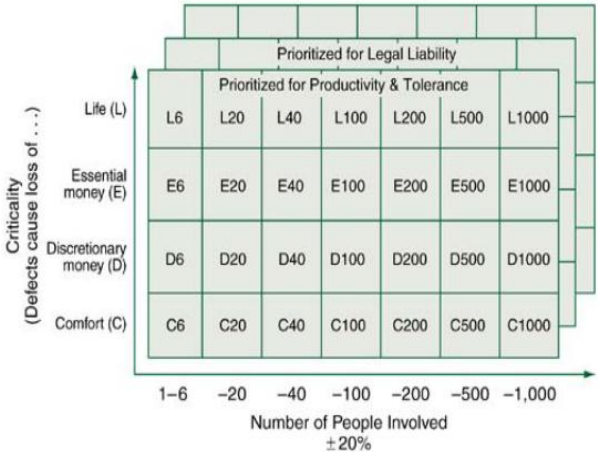

Fig 8 The Family of Crystal Methods

V. PROBLEM SOLVED BY AGILE

Here are reasons to apply agile development principles and practices. [16]

a) *Revenue*: The iterative nature of agile development means features are delivered incrementally, enabling some benefits to be realized early as the product continues to develop.

b) *Speed-to-market*: Research suggests about 80% of all market leaders were first to market. As well as the higher revenue from incremental delivery, agile development philosophy also supports the notion of early and regular releases, and 'perpetual beta'.

c) *Quality*: A key principle of agile development is that testing is integrated throughout the lifecycle, enabling regular inspection of the working product as it develops. This allows the product owner to make adjustments if necessary and gives the product team early sight of any quality issues.

d) *Visibility*: Agile development principles encourage active 'user' involvement throughout the product's development and a very cooperative collaborative approach. This provides excellent visibility for key stakeholders, both of the project's progress and of the product itself, which in turn helps to ensure that expectations are effectively managed.

e) *Risk Management*: Small incremental releases made visible to the product owner and product team through its development help to identify any issues early and make it easier to respond to change. The clear visibility in agile development helps to ensure that any necessary decisions can be taken at the earliest possible opportunity, while there's still time to make a material difference to the outcome.

f) *Flexibility / Agility*: In traditional development projects, we write a big spec up-front and then tell business owners how expensive it is to change anything, particularly as the project goes on. In fear of scope creep and a never-ending project, we resist changes and put people through a change control committee to keep them to the essential minimum. Agile development principles are different. In agile development, change is accepted. *In fact, it's expected*. Because the one thing that's certain in life is change. Instead the timescale is fixed and requirements emerge and evolve as the product is developed. Of course for this to work, it's imperative to have an actively involved stakeholder who understands this concept and makes the necessary trade-off decisions, trading existing scope for new.

g) *Cost Control*: The above approach of fixed timescales and evolving requirements enables a fixed budget. The scope of the product and its features are variable, rather than the cost.

h) *Business Engagement/Customer Satisfaction*: The active involvement of a user representative and/or product owner, the high visibility of the product and progress, and the flexibility to change when change is needed, create much better business engagement and customer satisfaction. This is an important benefit that can create much more positive and enduring working relationships.

i) *Right Product*: Above all other points, the ability for agile development requirements to emerge and evolve, and the ability to embrace change (with the appropriate trade-offs), the team build the right product. It's all too common in more traditional projects to deliver a "successful" project in IT terms and find that the product is not what was expected, needed or hoped for. In agile development, the emphasis is absolutely on building the right product.

j) *More Enjoyable*!: The active involvement, cooperation and collaboration make agile development teams a much more enjoyable place for most people. Instead of big specs, we discuss requirements in workshops. Instead of lengthy status reports, we collaborate around a task-board discussing progress. Instead of long project plans and change management committees, we discuss

what's right for the product and project and the team is empowered to make decisions. In my experience this makes it a much more rewarding approach for everyone. In turn this helps to create highly motivated, high performance teams that are highly cooperative.

VI. FUTURE WORK

As we know that agile is best software development method but till now agile is not so popular for government projects and defense projects. In Forrester Research surveys of 1000+ IT professionals in 2009 and in 2010, showed a decline in the use of "Traditional" development methodologies, and a rise in adoption of Agile. About 40% of organizations reported using one or another of the Agile family of methodologies [12,13]. Studies show a wide range of positive effects of Agile: 5% to 61% reductions in cost, 24% to 58% reductions in development time, and 11% to 83% reductions in product defects [15].

Despite these results, in the government, and particularly the defense sector, Agile is resisted. A 2010 study of Agile in defense software acquisition [14] found that Agile was only likely to be used in two circumstances: 1) the program is urgent and mission critical for combat forces and has the clout to get a waiver from the normal process, or 2) the program is failing and likely to be canceled. So in future we can analyze the reason behind this discrimination and what can we do to solve this problem?

VII. CONCLUSION

this paper is written after studying many papers on agile methodology. This paper explains different types of SDLCs like Waterfall, incremental, spiral, v-model. This paper also defined problem associated with these traditional SDLCs. In this paper we explained AGILE method its different–different approaches namely eXtreme Programming, SCRUM, Adaptive Software Development, Crystal. And list down the reasons to apply agile development principles and practices. To explain this paper there is a table which provides full review of Agile compared with traditional method.

Traditional Methodology	**Agile methodology**
Requirements are clear at the inception of the project.	Requirements are not clear at the inception of the project.
Limited communication, more stress in documentation.	More communication allows developing a product over short period of time, lesser importance on the documentation.
Elaborate process should be followed.	Limited process involved.
Time consuming as they develop the complete product at a single cycle.	Iterative model allow developing easily in a short span of time.
No scrums calls and stand up calls.	Scrums calls are stand up calls at a regular interval to track the progress and get feedback from the clients.
Time consuming to understand, develop, test and defect fixing.	Save a lot of time due to frequent communication with stack holders, iterative development and proper feedback.
Testing happens only after the completion of the development.	Testing team work in parallel with the development team which helps to find the defect as soon as possible.
Automation is not a usual practice.	Continuous automated testing which ensures better quality.

Table. 1 Traditional Vs. Agile methodology

REFERENCES

[1] M. A. Awad. A comparison between agile and traditional software development methodologies. Master's thesis, The University of Western Australia, 2005.
[2] K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. C. Martin, S. Mellor, K. Schwaber, J. Sutherland, and D. Thomas. The twelve principles of agile software, 2001.
[3] K. Beck. Embracing change with extreme programming. Computer, 32(10):70–77, Oct. 1999.
[4] A. Cockburn. Agile Software Development. Addison- Wesley Professional, 2001.
[5] R. E. Jeffries, A. Anderson, and C. Hendrickson. Extreme Programming Installed. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2000.
[6] J. Highsmith. Retiring lifecycle dinosaurs. Software Testing & Quality Engineering (STQE), pages 22–28, 2000.
[7] K. Schwaber. Controlled chaos: Living on the edge. American Programmer 9, 5:10–16, 1996.
[8] K. Schwaber. Against a sea of troubles: Scrum software development. Cutter, 13:34–39, 2000.
[9] J. Sutherland. Agile can scale: Inventing and reinventing scrum in five companies. Cutter IT Journal, 14:5–11, 2001.
[10] http://www.agilealliance.org/the-alliance/the-agile-manifesto
[11] http://agilemanifesto.org/principles.html
[12] D. West, et al. (2010). "Agile Development: Mainstream Adoption Has Changed Agility." *Forrester Research.*
[13] D. West et al. (2011) "Water-Scrum-Fall Is The Reality Of Agile For Most Organizations Today." *Forrester Research.*
[14] M. Lapham, et al. (2010) "Considerations for Using Agile in DoD Acquisition." Carnegie Mellon, Software Engineering Institute: April 2010, Technical Note CMU/SEI-2010-TN -002.
[15] Rico, D. F. (2008). "What is the return of investment (ROI) of agile methods?"
[16]http://www.allaboutagile.com/10-good-reasons-to-do-agile-development/#sthash.Jtxdo8tU.dpuf
[17] J. Highsmith. Agile Software Development Ecosystems.Addison-Wesley Professional, 2002.
[18]http://en.wikipedia.org/wiki/Systems_Development_Life_Cycle
[19] http://en.wikipedia.org/wiki/Agile_software_development
[20] http://agilemanifesto.org
[21] http://www.agilealliance.org
[22]Lean Software Development: An Agile Toolkit for Software Development Managers - by Mary Poppendieck, Tom Poppendieck, Ken Schwaber .