International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Index Copernicus Value (2013): 6.14 | Impact Factor (2013): 4.438

An Overview of Agile Software Development Process

Abhilasha Yadav

Sri Satya Sai Institute of Science & Technology, Sehore

Abstract: ***We all know that in recent modern era software is the basic requirement for operating almost every digital machine. So that we have to think about software and about creation of software which is efficient to fulfill the requirement and consume less time. And after studying about many software development method there quality and loopholes the final outcome is documented in this paper. This paper is all about agile, concept behind agile, range of agile, how agile achieve quality under time pressure and changing requirement environment? The Advantages and disadvantages of agile.***

Keyword: Agile methodology, The Agile Manifesto, Agile quality technique

1. Introduction

Software plays a very crucial role in numerous numbers of areas of technology and business world, as it is driven extensively both by the individuals and the companies either as a single main application or as a part of an aggregate project in order to ease the level of effort, raise functionality and consistency of the work by computerizing procedures and enabling services.
Due to the constant development of information systems, the increasing demand in the field and as the business has taken different forms over the years, different software development methods and models have been invented and used over the last five decades in order to facilitate the development processes.

With the aforementioned qualities, some of the traditional methods have turned into highly documentation-oriented ways of development and such models strictly requiring and limiting developers to follow and apply certain processes. As a counter-reaction to those problems the traditional methods had brought, the understanding of the significance of human factor, importance of collaboration and communication between the team and the customers, and the value of ability to respond to changes has begun to arise in the software industry. This leaded the new Agile Methods be formulated and applied over the last two decades, in order to overcome the shortcomings of the traditional methods.

The reality is that agile methods have gained tremendous acceptance in the commercial arena since late 90s because they accommodate volatile requirements, focus on collaboration between developers and customers, and support early product delivery. Two of the most significant characteristics of the agile approaches are[10]:
1. They can handle unstable requirements throughout the development lifecycle
2. They deliver products in shorter timeframes and under budget constraints when compared with traditional development methods.

Compare the quality assurance techniques of agile and traditional software development processes. In three steps:
I. build a complete outline of the traditional model including its supporting processes,
II. Identify those practices within agile methods that purport to ensure software quality,
III. Determine the similarities and differences between agile and traditional software quality assurance techniques.

By applying such an approach, we can systematically investigate how agile methods integrate support for software quality within their life cycle.

This paper contain ^ parts, part 1 introduction, Part 2 about Agile software development, part 3 Agile methods Quality techniques, part 4 Advantage & Disadvantage , part 5 Future Work and part 6 Conclusion.

2. Agile Software Development

Agile software development is a group of software development methods in which requirements and solutions evolve through collaboration between self-organizing, cross-functional teams. It promotes adaptive planning, evolutionary development, early delivery, continuous improvement and encourages rapid and flexible response to change. It is a conceptual framework that focuses on frequently delivering small increments of working software.

To be able to fully understand the concept, it is helpful to first describe the meaning of the term “agility” from different perspectives. The term agility was defined well in Agile Competitors and Virtual Organizations, in fact for flexible manufacturing, as “Agility is dynamic, context-specific, aggressively change-embracing, and growth-oriented. It is not about improving efficiency, cutting costs, or battening down the business hatches to ride out fearsome competitive “storms”. It is about succeeding and about winning: about succeeding in emerging competitive arenas, and about winning profits, market share, and customers in the very center of the competitive storms many companies now fear.”[2][7]

So-called *lightweight* agile software development methods evolved in the mid-1990s as a reaction against the *heavyweight* waterfall-oriented methods, which were characterized by their critics as being heavily regulated, regimented, micromanaged and over-incremental approaches to development.

Proponents of lightweight agile methods contend that they are returning to development practices that were present early in the history of software development.

Early implementations of agile methods include Rational Unified Process (1994), Scrum (1995), Crystal Clear, Extreme Programming (1996), Adaptive Software Development, Feature Driven Development (1997), and Dynamic Systems Development Method (DSDM) (1995). These are now collectively referred to as agile methodologies, after the Agile Manifesto was published in 2001.[7]

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[20]:

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

In 2002, along with the four main values Agile Alliance has published The Twelve Principles behind the Agile Manifesto that further explicate what it is to be Agile. These principles are as follows[19]:

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.

Agile methodologies include:

- Extreme Programming
- Agile Modeling
- SCRUM
- Test-driven development (TDD)
- Feature-driven development (FDD)

a) *Extreme Programming (XP)*

Extreme Programming was introduced by Kent Beck in 2000. XP offers a number of practices, values and principles for software development project. Extreme Programming was in fact targeted especially at small co-located teams developing non-critical products [6][12]. Currently, however, it has been used by companies of all different sizes and industries worldwide.

1. *Architecture Spike*-The holy grail of XP is the delivery of working software within a short time frame. This exclusive focus on functionality could easily turn the project into a 'Feature Factory', with the risk of accumulating architectural deficiencies and technical debt that has an impact on performance of the software and overall product quality. This risk can be mitigated by incorporation of a sound architectural basis for the system under development, applying 'Architecture Spikes'.[2][12]
2. *Release Planning*-Planning and estimating in the agile world depend on a single key metric: the development team's velocity, which describes how much work the team can get done per iteration. (We describe velocity in detail separately.) Given a team's known velocity for its last project (if it is known), a release plan represents how much scope that team intends to deliver by a given deadline. [2][12]. The goal of initial release planning is to estimate roughly which features will be delivered by the release deadline (presuming the deadline is fixed), or to choose a rough delivery date for a given set of features (if scope is fixed). We use this information to decide whether or not the project will produce enough ROI to at least pay for itself, and therefore whether or not we should proceed.
3. *Iteration*-Iteration is the act of repeating a process with the aim of approaching a desired goal, target or result. Each repetition of the process is also called an "iteration", and the results of one iteration are used as the starting point for the next iteration. The pentagon on the right also is a good example of how iteration relates to recursion. Although iteration is used, for example, to parse a linked list, recursion is required when we step up to binary trees. The pentagon demonstrates both. [2][12]
4. *Acceptance testing* -In engineering and its various subdisciplines, acceptance testing is a test conducted to determine if the requirements of a specification or contract are met. It may involve chemical tests, physical tests, or performance tests.In systems engineering it may involve black-box testing performed on a system (for example: a piece of software, lots of manufactured mechanical parts, or batches of chemical products) prior to its delivery.[1] Software developers often distinguish acceptance testing by the system provider from acceptance testing by the customer (the user or client) prior to accepting transfer of ownership. In the case of software, acceptance testing performed by the customer is known as user acceptance testing (UAT), end-user testing, site (acceptance) testing, or field (acceptance) testing. [2][12]

5. *Small Release planning*-The development team needs to release iterative versions of the system to the customers often. Some teams deploy new software into production every day. At the very least you will want to get new software into production every week or two. At the end of every iteration you will have tested, working, production ready software to demonstrate to your customers. The decision to put it into production is theirs. [2][12]
   The release planning meeting is used to plan small units of functionality that make good business sense and can be released into the customer's environment early in the project. This is critical to getting valuable feedback in time to have an impact on the system's development. The longer you wait to introduce an important feature to the system's users the less time you will have to fix it.

The programming principles that are encouraged by XP are:

- Its simplicity and flexibility to reduce maintenance costs of the software.
- The intensive and robust testing mechanism that reduces the number of defects after delivery.
- Embraces changes, during the development process.
- Moreover, encourages creating high quality code [12]

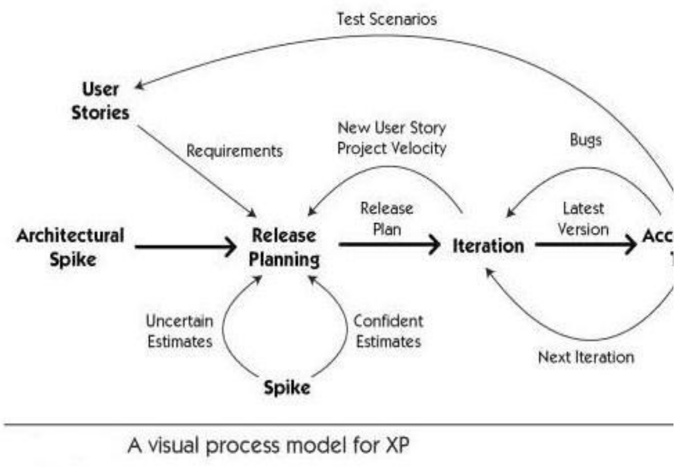


Figure 1: Extreme Programming

b) Agile Modeling (AM)

Modeling is important and enables software developers to think about complex issues prior to its implementation. Agile Modeling (AM) was established by Scott Ambler in 2002. It is a collection of values, principles, and practices for modeling software for development project in an effective and light-weight manner [10].

Humility means to accept that you may not know everything; others may also provide useful contribution to the project [10].
Again, the principles of AM are quite similar to those of XP, such as assuming simplicity, accepting changes, incrementally of the system, and rapid feedback. In addition to these principles, AM include the knowledge of the purpose for modeling; having multiple effective models; the content is more important than the representation; keeping open and honest communication between parties involved in the development process; and finally, to focus on the quality of the work [10].
The practices of AM have some commonalities with those of XP, too. AM practices highlight on active stakeholder participation focus on group work to create the suitable models; apply the appropriate artifact as UML diagrams; verify the correctness of the model, implement it and show the resulting interface to the user; model in small increments; create several function models in parallel; apply modeling standards; and other practices [10].

Agile Model Driven Development (AMDD) is the agile version of model driven development. To apply AMDD, an overall high level model for the whole system is created at the early stage of the project. During the development iterations, the modeling is performed per iteration, along with other methodologies, such as Test Driven Development (TDD), and Extreme Programming (XP), to get the best results [10].

AM basically creates a mediator between rigid methodologies and lightweight methodologies, by suggesting that developers communicate architectures through applying its practices to the modeling process.

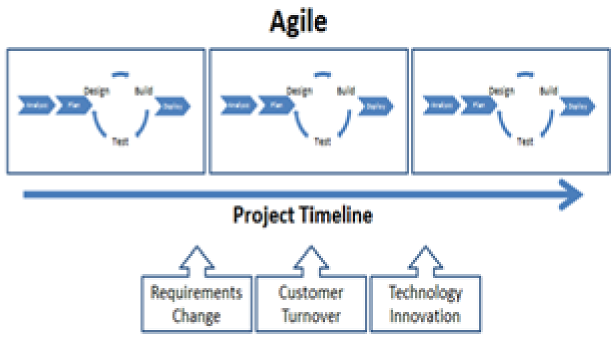


Figure 2: Agile Modeling

c) SCRUM

SCRUM methodology was introduced by Ken Swaber in 1995 and was practiced before the announcement of Agile Manifesto. Later, it was included into agile methodology. SCRUM is used with the objective of simplifying project control through simple processes, easy to update documentation and higher team iteration over exhaustive documentation [1][9].
Scrum is a management framework for incremental product development using one or more cross-functional, self-organizing teams. It provides structure of roles, meetings, rules, and artifacts. Teams are responsible for creating and adapting their processes within this framework. Scrum uses fixed-length iterations, called Sprints.

1. *Initiate* - This phase includes the processes related to initiation of a project: Create Project Vision, Identify Scrum Master and Stakeholder(s), Form Scrum Team, Develop Epic(s), Create Prioritized Product Backlog, and Conduct Release Planning.[15]
2. *Plan and Estimate* - This phase consists of processes related to planning and estimating tasks, which include Create User Stories, Approve, Estimate, and Commit User Stories, Create Tasks, Estimate Tasks, and Create Sprint Backlog.[15]

3. *Implement* - This phase is related to the execution of the tasks and activities to create a project's product. These activities include creating the various deliverables, conducting Daily Standup Meetings, and grooming (i.e., reviewing, fine-tuning, and regularly updating) the Product Backlog at regular intervals. [15]
4. *Review and Retrospect* - This phase is concerned with reviewing the deliverables and the work that has been done and determining ways to improve the practices and methods used to do project work.[15]
5. *Release* - This phase emphasizes on delivering the Accepted Deliverables to the customer and identifying, documenting, and internalizing the lessons learned during the project.[15]

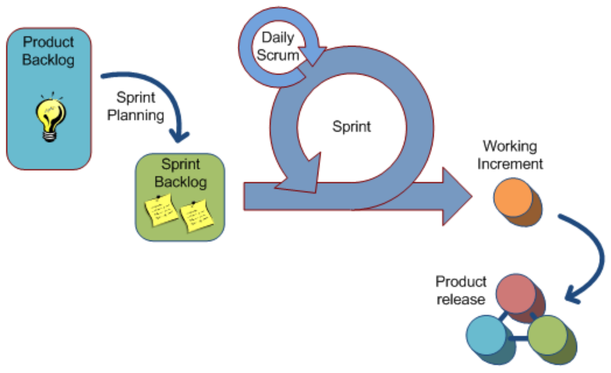


Figure 3: SCRUM Process Stages

Schwaber lists the key principles of SCRUM as follows[21]:
- 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...)

d) Test-driven development (TDD)

Test-driven development (TDD) is a software development process that relies on the repetition of a very short development cycle: first the developer writes an (initially failing) automated test case that defines a desired improvement or new function, then produces the minimum amount of code to pass that test, and finally refactors the new code to acceptable standards. Kent Beck, who is credited with having developed or 'rediscovered' the technique, stated in 2003 that TDD encourages simple designs and inspires confidence.[16]

Test-driven development is related to the test-first programming concepts of extreme programming, begun in 1999,[3] but more recently has created more general interest in its own right.[4]. It includes:

1. *Add a test*-In test-driven development, each new feature begins with writing a test. To write a test, the developer must clearly understand the feature's specification and requirements. The developer can accomplish this through use cases and user stories to cover the requirements and exception conditions, and can write the test in whatever testing framework is appropriate to the software environment.
2. *Run all tests and see if the new one fails*-This validates that the test harness is working correctly, that the new test does not mistakenly pass without requiring any new code, and that the required feature does not already exist. This step also tests the test itself, in the negative: it rules out the possibility that the new test always passes, and therefore is worthless.
3. *Write some code*-The next step is to write some code that causes the test to pass. The new code written at this stage is not perfect and may, for example, pass the test in an inelegant way. That is acceptable because it will be improved and honed in Step 5.

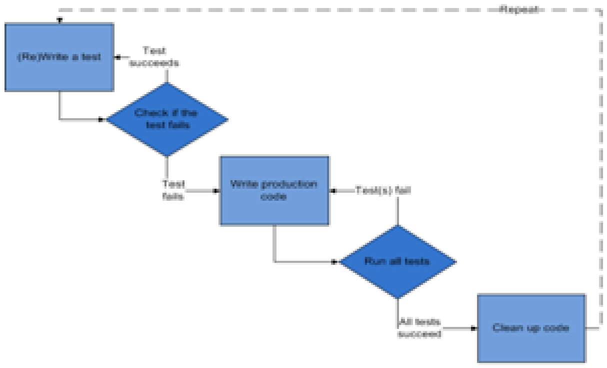


Figure 4: Test-driven development

4. *Run tests*-If all test cases now pass, the programmer can be confident that the new code meets the test requirements, and does not break or degrade any existing features. If they do not, the new code must be adjusted until they do.
5. *Refactor code*-The growing code base must be cleaned up regularly during test-driven development. New code can be moved from where it was convenient for passing a test to where it more logically belongs. Duplication must be removed. Object, class, module, variable and method names should clearly represent their current purpose and use, as extra functionality is added. As features are added, method bodies can get longer and other objects larger.

e) Feature-driven development (FDD)

Feature-driven development (FDD) is an iterative and incremental software development process. It is one of a number of lightweight or Agile methods for developing software. FDD blends a number of industry-recognized best practices into a cohesive whole. These practices are all driven from a client-valued functionality (feature) perspective. Its main purpose is to deliver tangible, working software repeatedly in a timely manner.[17]

FDD is a model-driven short-iteration process that consists of five basic activities.

1. *Develop overall model*-The FDD project starts with a high-level walkthrough of the scope of the system and its context. Next, detailed domain models are created for each modeling area by small groups and presented for peer review. One of the proposed models, or a combination of them, is selected to become the model for each domain area. Domain area models are progressively merged into an overall model.

2. *Build feature list*-The knowledge gathered during the initial modeling is used to identify a list of features, by functionally decomposing the domain into subject areas. Subject areas each contain business activities, and the steps within each business activity form the basis for a categorized feature list. Features in this respect are small pieces of client-valued functions expressed in the form "<action> <result> <object>", for example: 'Calculate the total of a sale' or 'Validate the password of a user'. Features should not take more than two weeks to complete, else they should be broken down into smaller pieces.

3. *Plan by feature*-After the feature list is completed, the next step is to produce the development plan; assigning ownership of features (or feature sets) as classes to programmers.

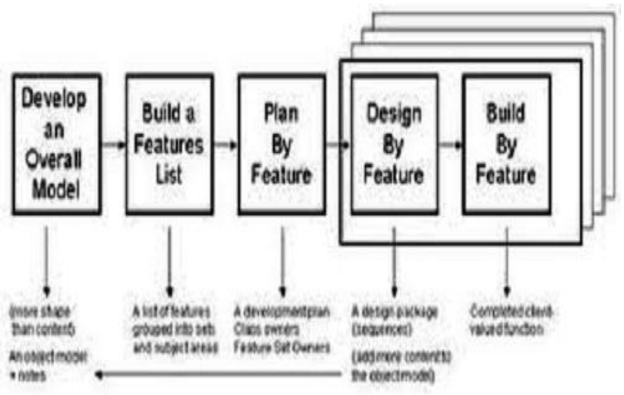

Develop an Overall Model
Build a Features List
Plan By Feature
Design By Feature
Build By Feature
(more shape than content)
An object model + notes
A list of features grouped into sets and subject areas
A development plan Class owners Feature Set Owners
A design package (sequences)
(add more content to the object model)
Completed client-valued function

Figure 5: Feature-driven development

4. *Design by feature*-A design package is produced for each feature. A chief programmer selects a small group of features that are to be developed within two weeks. Together with the corresponding class owners, the chief programmer works Together with the corresponding class owners, the chief programmer works out detailed sequence diagrams for each feature and refines the overall model. Next, the class and method prologues are written and finally a design inspection is held.

1. *Build by feature*-After a successful design inspection as per feature activity to produce a completed client-valued function (feature) is planned. The class owners develop the code for their classes. After a unit test and a successful code inspection, the completed feature is promoted to the main build.

Table 1

	Agile quality technique						
Methodology	Feature Driven	Iterative-Incremental	Refactoring	Micro-Optimizing	Customer Involvement	Team Dynamics	Continuous Integration
Scrum	X	X		X	X	X	
XP	X	X	X	X	X	X	
TDD	X	X	X				X
FDD	X	X					
Crystal	X	X		X	X		

Agile software development can be considered as merely "a collection of practices, a frame of mind", it is difficult to tell whether a company's process model is defined as Agile. Some may choose to follow those Agile beliefs loosely while others may employ a strict Agile system. Therefore it is important to distinguish companies with different levels of "agility" in order to properly analyze the effectiveness of the agile system.
Agile methodology will be defined as [7][2]
1. Follows all the Agile principles and strict practices, similar to Extreme Programming and Scrum.
2. Agile development may have different effects on a company depending on its stage and size. The cost of implementing the Agile methodology and the benefits vary as the company grows.
3. We must first discuss the scope of the metrics that we are using to determine the effectiveness of a process model. The metrics used in this discussion are cost, time, quality and scope as they apply to a startup.

3. Agile Methods Quality Techniques

Agile methods include many practices that have QA potential. By identifying these practices and comparing them with QA techniques used in the waterfall model, we can analyze agile methods QA practices. System metaphor is used instead of a formal architecture. It presents a simple shared story of how the system works; this story typically involves a handful of classes and patterns that shape the core flow of the system being built. There are two main purposes for the metaphor. The first is communication. It bridges the gap between developers and users to ensure an easier time in discussion and in providing examples. The second purpose is that the metaphor contributes to the team's development of software architecture.

Agile methods move into the development phase very quickly. Although this kind of development style renders most separate static techniques on early phase artifact unsuitable, code makes dynamic techniques useful and available very early. Also developers are more responsible for quality assurance compared with having a separate QA team and process. This allows more integration of QA into the development phase. Small releases also bring customer feedback for product validation frequently and requirements verification. The QA techniques for agile methods are based on:

- Applying dynamic QA techniques as early as possible (e.g. TDD, acceptance testing)
- Moving more QA responsibility on to the developer (e.g. code inspection in peer/pair programming, refactoring, collective code ownership, coding standards)
- Early product validation [10] (e.g. customer on site, acceptance testing, small release, continuous integration)

a) System Metaphor

Metaphor is a way to express meaning in a condensed manner by referring to qualities of known entities. Metaphors are useful because they are efficient. The system metaphor is a story that everyone: customers, programmers, and managers, can tell about how the system works. We seek a system metaphor for several reasons: common vision, shared vocabulary, generativity, and architecture [10]. It presents a simple shared story of how the system works, this story typically involves a handful of classes and patterns that shape the core flow of the system being built. The idea of using a System metaphor to facilitate communication works toward revealing the reality of the team towards its task. For a team starting out, metaphors are a comfortable and flexible starting point and they leave open the chance to use the metonymy that patterns provide [9]. System metaphor does not seem to address bigger architectural issues such as if the system should even be implemented using objects, the hardware component of architecture, the process and inter-process communication component of architecture, or the separation of the system into layers and/or components [9].System metaphor is helpful for communication between customer and developer. It helps the agile development team in architectural evaluation by increasing communication between team members and users. So enhance maintainability, efficiency, reliability and flexibility.

b) Architectural Spike

An architectural spike is technical risk reduction techniques popularized by Extreme Programming (XP) where write just enough code to explore the use of a technology or technique that you're unfamiliar with. For complicated spikes architecture owners will often pair with someone else to do the spike [4][2]. It intends to identify areas of maximum risk, to get started with estimating them correctly. Agile projects are designed for iteration at a time. It is a thin slice of the entire application built for the purpose of determining and testing a potential architecture.

c) Onsite Customer Feedbacks

Onsite customer is one of the most practices in most agile projects that help the developers refine and correct requirements throughout the project by communicating. Customer are only involved during requirement collecting in traditional software developments but they are directly involved in agile methodology. All phases of agile project require communication with the customer, preferably face to face, on site. It's best to simply assign one or more customers to the development team. A real customer must sit with the team, available to answer questions, resolve disputes, and set small-scale priorities. Agile is intended to improve software quality and responsiveness to changing customer requirements. As a type of agile software development it advocates frequent releases in short development cycles, which is intended to improve productivity and introduce checkpoints where new customer requirements can be adopted.

d) Refactoring

Refactoring is a disciplined technique for restructuring an existing body of code, altering its internal structure without changing its external behavior. Its heart is a series of small behavior preserving transformations. Each transformation (called a 'refactoring') does little, but a sequence of transformations can produce a significant restructuring. Since each refactoring is small, it's less likely to go wrong. The system is also kept fully operational after each small refactoring. Practically refactoring means making code clearer and cleaner and simpler and well designed. It can reduce the chances that a system can get seriously broken during the restructuring [11][2]. So refactoring reduces the probability of generating errors for the period of developments, hence improve software quality factors such as efficiency, reliability, intra-operability and interoperability, testability.

e) Pair Programming

Pair programming is a technique in which two programmers or engineers work together at one workstation. One writes code while the other, the observer, reviews each line of code as it is typed in. The best way to pair program is to just sit side by side in front of the monitor. Slide the key board and mouse back and forth. Both programmers concentrate on the code being written and continuously collaborating on the same design, algorithm, code or test [6][12]. The two programmers switch roles frequently. While reviewing, the observer also considers the strategic direction of the work, coming up with ideas for improvements and likely future problemsto address. This procedure increases software quality without impacting time to deliver. Pair programming can improve design quality factors such as correctness, verifiability, and testability and reduce defects [12][6].

f) Stand-up-Meeting

Stand-up-meeting is the most important practice in agile methods. It increases the communication between term members and developers. A stand-up meeting (or simply "stand-up") is a daily team meeting held to provide a status update to the team members. Communication among the entire team is the purpose of the stand-up- meeting. The meeting is usually held at the same time and place every working day. All team members are encouraged to attend, but the meetings are not postponed if some of the team members are not present.[1][15].

4. The Advantages and Disadvantages of agile

Advantages

- It is a very realistic approach to software development
- It Promotes teamwork and cross training.

- Its Functionality can be developed rapidly and demonstrated.
- Its Resource requirements are minimum.
- It is Suitable for fixed or changing requirements
- It delivers early partial working solutions.
- It is Good model for environments that change steadily.
- Its Minimal rules, documentation easily employed.
- Enables concurrent development and delivery within an overall planned context.
- Little or no planning required
- Easy to manage
- Gives flexibility to developers
- Customer satisfaction by rapid, continuous delivery of useful software.
- People and interactions are emphasized rather than process and tools. Customers, developers and testers constantly interact with each other.
- Working software is delivered frequently (weeks rather than months).
- Face-to-face conversation is the best form of communication.
- Close, daily cooperation between business people and developers.
- Continuous attention to technical excellence and good design.
- Regular adaptation to changing circumstances.
- Even late changes in requirements are welcomed

Disadvantages

- Not suitable for handling complex dependencies.
- More risk of sustainability, maintainability and extensibility.
- An overall plan, an agile leader and agile PM practice is a must without which it will not work.
- Strict delivery management dictates the scope, functionality to be delivered, and adjustments to meet the deadlines.
- Depends heavily on customer interaction, so if customer is not clear, team can be driven in the wrong direction.
- There is very high individual dependency, since there is minimum documentation generated.
- Transfer of technology to new team members may be quite challenging due to lack of documentation.
- In case of some software deliverables, especially the large ones, it is difficult to assess the effort required at the beginning of the software development life cycle.
- There is lack of emphasis on necessary designing and documentation.
- The project can easily get taken off track if the customer representative is not clear what final outcome that they want.
- Only senior programmers are capable of taking the kind of decisions required during the development process. Hence it has no place for newbie programmers, unless combined with experienced resources.

5. Future Work

A possible area for future research is the analysis of the effects of process models on mobile-centric startups. Practices such as continuous integration in Agile or continuous deployment in Lean Startup become nearly impossible in a startup with heavy focus on mobile development. The process models that focus heavily on the ability to integrate and deploy continuously or split testing may not be effective for mobile-centric startups. In this new era in which mobile development is becoming more and more popular, perhaps a new process model is required.

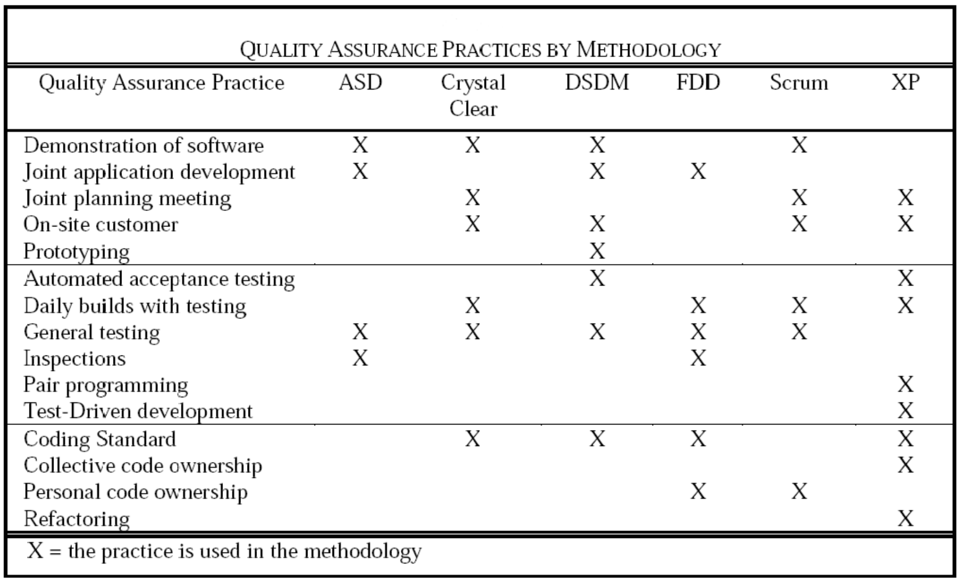
6. Conclusion

The Agile Manifesto is considered as a milestone for the development of new agile methodologies, since it provided a general base with its four values pointing out the most crucial facts of a software development process.

The process or role oriented traditional methods have difficulties, since individuals are not, in fact, replaceable; especially where the individuality is important as in software development. Furthermore, rather than focusing on strict and long-term plans, Agile Software Development focuses on responding to changes and on the working software with less documentation. Even though some agile practices are not new, agile methods themselves are recent and have become very popular in industry. There is an important need for developers to know more about the quality of the software produced. Developers also need to know how to revise or tailor their agile methods in order to attain the level of quality they require. The conclusion we draw here is:

1. Agile methods do have practices that have QA abilities, some of them are inside the development phase and some others can be separated out as supporting practices
2. The frequency with which these agile QA practices occur is higher than in a waterfall development
3. Agile QA practices are available in very early process stages due to the agile process characteristics.

Table 2: Provides a comparison among different discussed methodologies and quality assurance factors obtained from them in different processes. Pragmatic Programming has been intentionally left blank due to its immature nature and unavailability in literature.



QUALITY ASSURANCE PRACTICES BY METHODOLOGY

Quality Assurance Practice	ASD	Crystal Clear	DSDM	FDD	Scrum	XP
Demonstration of software	X	X	X		X	
Joint application development	X		X	X		
Joint planning meeting		X			X	X
On-site customer		X	X		X	X
Prototyping			X			
Automated acceptance testing			X			X
Daily builds with testing		X		X	X	X
General testing	X	X	X	X	X	
Inspections	X			X		
Pair programming						X
Test-Driven development						X
Coding Standard		X	X	X		X
Collective code ownership						X
Personal code ownership				X	X	
Refactoring						X

X = the practice is used in the methodology

Reference

[1] Cristal. M,Wildt. D. and Prikladnicki. R (2008) Usage of SCRUM Practices within a Global Company. In IEEE International Conference on Global Software Engineering, pp 222-226.
[2] Erickson. J, Lyytinen. K and Siau. K, (2005), Agile Modeling, Agile Software Development, and Extreme Programming: The State of Research. In Journal of Database Management, 16(4), pp 88-100.
[3] McCall,J.A., Richards,P. K., & Walters, G. F., Factors in Software Quality, Vols.1, 2 and 3, National Technical Information Service, 1977
[4] Ambler,S., Quality in an Agile World, Software Quality Professional, Vol. 7, No. 4, pp. 34-40,2005
[5] "IEEE guide for software quality assurance planning", IEEE Std. 730.1-1995, 10 April 1996.
[6] Beck K., "Extreme Programming Explained: Embrace Change", Addison- Wesley, 2000.
[7] Cockburn A., "Agile Software Development", Addison-Wesley, 2001.
[8] George, B., Williams L., "An Initial Investigation of Test-Driven development in Industry", Proceedings of the ACM symposium on applied computing, March 2003.
[9] Schwaber K, Beedle M., "Agile Software Development With Scrum", Upper Saddle River, NJ: Prentice-Hall, 2002.
[10] Ming Huo, June Verner, Liming Zhu, Muhammad Ali Babar, Software Quality and Agile Methods, Proceedings of the 28th Annual *International Computer Software and Applications Conference (COMPSAC'04)*
[11] www.agilemodeling.com
[12] www.extremeprogramming.org
[13] http://en.wikipedia.org/Crystal_Clear
[14] http://en.wikipedia.org/agile_molding
[15] http://en.wikipedia.org/SCRUM
[16] http://en.wikipedia.org/TDD
[17] http://en.wikipedia.org/FDD
[18] http://www.agilealliance.org/the-alliance/the-agile-manifesto
[19] http://agilemanifesto.org/principles.html
[20] http://www.agilemanifesto.org