International Journal of Software Engineering and Computer Systems (IJSECS)
ISSN: 2289-8522, Volume 5 Issue 2, pp. 1-14, August 2019

https://doi.org/10.15282/ijsecs.5.2.2019.1.0057

COMPARISON OF TRADITIONAL AND AGILE SOFTWARE DEVELOPMENT METHODOLOGY: A SHORT SURVEY

Sarang Shaikh, Sindhu Abro
{sarang.msse17, sindhu.mscs17}@iba-suk.edu.pk
Department of Computer Science, Sukkur IBA University
Main Airport Road, Sukkur, Pakistan.

ABSTRACT

Software Development Methodologies (SDM) are used for every activity performed on a software product from initiation to maintenance. There are a variety of software development methodologies (waterfall, spiral and iterative) that are available to develop software products. One of the key challenges faced by the software developer is the selection of SDM in a software product. No single methodology is ideal to work effectively in all scenarios. Therefore, software product features play an important role in the SDM selection. This paper aims to explain different features, characteristics, critical practices, advantages, disadvantages of different methodologies related to the software product. We have used six models including waterfall, unified process, spiral, extreme programming, scrum, and feature-driven development. This paper also summarized the limitations and cost control factors of SDM while developing software products.

Keywords: Software Development Life Cycle (SDLC), Traditional Methodology, Agile Methodology, Software Development, Software Engineering.

1. INTRODUCTION

Someone only says for software, that our modern world depends on it. To support this, we discuss that in nearly past year's software development has become a difficult, challenging and important activity of the modern world. Software products today are somehow a picture of human ideas. So, the end-product is a representation of thoughts presented in binary codes other than physical quantity. That's why different techniques are required to produce such type of intangible products with higher quality, reduced time and cost of development (Leffingwell, 2010).

In the starting days of development, software developed was without a specific plan, the only listed and then implemented. As the level of thinking and technology increases, the old strategies started slowing down. Soto achieves three parameters for software products (quality, cost and time) software project management related persons have developed different techniques named "Software Development Methodologies/ Framework". The need to improve these is to achieve the best software products and to map, maintain and control these products as a general product.

At that time these frameworks were for small-scale products, but as time passes, the level and value of software products become larger and results in increases in the complexity and failure of development methodologies at a higher rate. The software

industry started to slow down due to such methodologies that adapted to speed up industry and quality of the software.
To resolve the above-discussed problems, two methodologies were developed
1) Traditional Methodologies and **2) Agile Methodologies** as shown in Figure 1.

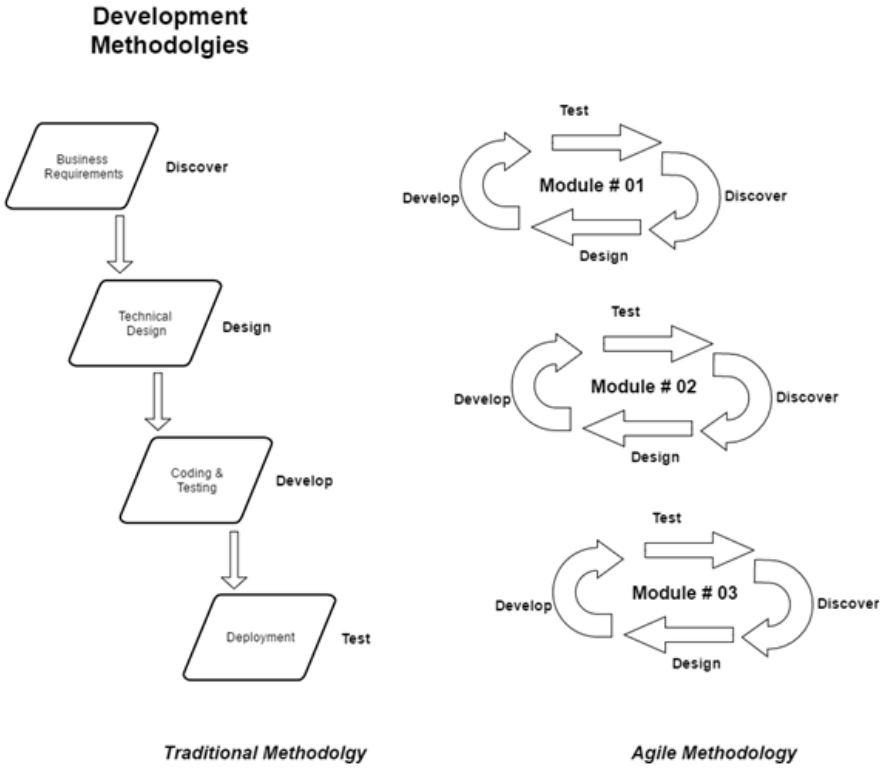

Figure 1 Traditional Vs Agile Development Methodology

The main purpose of our research is to present a comparative study between traditional and agile software development methodologies based on key features, implementation limitations, cost estimations success factors, and implementation issues. This study will help to select the most appropriate software development methodology for a specific software development project.

2. STATE OF THE ART

While discussing software development methodologies, it will be the best approach to categorizing those methodologies into two broad categories. The one is Traditional, and the other one is the Agile methodology. From now on the various models will be discussed below, after that, the multiple factors will be discussed which are quite important for choosing a method to implement. From those discussions, we will conclude that in which scenarios such as multiple team size, cost and budget issue and also time period, which methodology will be useful so that it can reduce cost and time as possible and guarantee a quality product. To check which methodology is best for the organization or a product, we are going to define three models from each methodology as well as advantages and disadvantages for comparison.

2.1 Traditional Methodology

It is the oldest method in the software industry since the 19th century; this approach is also known as Sequential Methodology. Like its name, "sequential" suggests a meaning that all the processes involved during the development of a product are a phase to phase-dependent on each other. Also, this methodology gives detailed documentation for each of the steps such as requirement engineering, design, coding, testing, deployment. It is a plan-driven methodology that starts after a detailed analysis and discussions; this approach is not useful when occurring of changes is entirely multiple times in the development (Awad, 2005).

2.2 Agile Methodology

As time passes and the software industry started growing in an efficient way the software analysts from all over the world combined in 2001, to have a talk on the future for software development methodologies. As all of these methodologies were supporting different software models discussed their standard features and concluded that there should be a combination of all of these, and the result is "Agile Methodology". As compared to traditional methodology, this method focuses on people, collaboration with customers, interaction with software rather than focusing on plans, processes, and tools.

Table 1 explains some of the major characteristics of each methodology (Awad, 2005).

Table 1 Major Characteristics of Methodology (Awad, 2005)

Traditional Methodology	**Agile Methodology**
Predictive Approach - This approach plans the whole project for an extended period or by keeping an eye on the future of the project. That plan is the basis for the construction of the system as it describes all the functions of the system, the role of each team member, cost and period for the development. All of this work has been done by prediction from earlier successful projects, whereas this plan sometimes also provides wrong assumptions in the future.	**Adaptive Approach -** This method is mainly for accepting changes in development. The agile methodology allows changes at every stage of development because it believes that changes in the requirements or anything are the best point to develop a customer satisfied product.
Comprehensive Documentation - In this methodology, documentation is the best key for development. It assumes that proper documentation should include - customer requirements, system requirements and all of the necessary information required for coding.	**Balancing Flexibility and Planning -** Although planning is the most important factor, planning for the whole project at a single time is not the right approach. Because there are many variables that change during the development, so to prepare for a short period is a good strategy or planning is such a way that you can quickly change your decisions or

	reverse your decisions according to the situation.
Process Oriented - There is a distinct process to do all of the work in the system, but not defined who will use this process as anyone can use this. This process describes the roles of managers, developers, analysts, testers, etc. but the procedure for these tasks is not clear.	**People-Oriented** - This methodology considers the people of the organization as the best part of the development rather than any process. The people involve managers, developers, designers, testers with high skills, talent and committed to the organization. It states that if people possess such qualities so they can quickly adapt and implement any process of development for any project. It helps to use multiple processes in the organization.

Continuing the discussion for comparison between these two methods, we have considered three software models from each methodology to compare different aspects and their roles in the software development life cycle. All these models are explained below with their basic definitions and development processes.

1) Waterfall Model

In the starting days of software engineering, the "code and fix" was the primary strategy applied by software methodologists to work on software projects, this means that first, you have to code the complete project and then to check for errors and fix it. As this approach evidently failed when there were large software projects. So, in 1970s Winston Royce proposed this methodology and called it the “waterfall model”. It is a sequential method in which the whole project is divided into seven stages or phases; the next step will only start when the previous one will be completed successfully and checked. All these measures contain some deliverables; a phase will end if and only if the required deliverables will be matched. This model is a baseline for some other software development life cycle models (SDLC).

2) Unified Process (UP) Model

It is a well-defined model, clearly explaining in a project what things need to be done, when and who will do. This model works using Unified Modelling Languages (UML), which means that all the phases, deliverables or outcomes are presented using UML diagrams (i-e: use cases, class diagrams, etc). It is a huge model that almost supports the development of all types of software products. This model works on three key features: 1) Incremental/Iterative, 2) Architecture focused 3) Use Case Determined. It is a component-based design, which creates such software systems that are easily understandable, supports software reuse and combines with Object-Oriented programming projects. The key feature of this model is that all the information is represented graphically. Also, the incremental feature supports the customer feedback, minimizes the risk and helps the developers.

3) Spiral Model

Barry Boehm developed this model, after a very detailed analysis of the waterfall model and Unified model, Boehm concluded that in the case of large software projects (i-e: government projects) these both models fail due to increasing of risks as well as changing of requirements due to significant time phase. So, this model is the combination of these two models as a primary focus or key focus on risk management of the product. This model involves some phases and iterations; the concept of stages is taken from the unified process whereas the idea of sequential repetition is taken from the waterfall model. All of these phases cover in a subsequent iteration, and by the end of the first iteration, our product is ready, and the customer feedback is demanded and checked. Based on those comments the second iteration starts and then another version of the product is released. As described above that, it is a risk-focused process model, due to cycles of this model risks can be easily found, resolved and then the particular sequential model like waterfall can be applied for development purpose (Leffingwell, 2010), (Awad, 2005), (Munassar & Govardhan, 2010), (Aitken & Ilango, 2013), (Kroll & Kruchten, 2003).

4) Extreme Programming (XP) Model

This process introduced in 1996, is a disciplined software development process. A lot of the research work is going in this process, and this method is taught in many of the software engineering courses in the educational institutes (Process, 2001). The biggest problems for which XP developed were fast-changing requirements from the customer side, so in the XP the focus is only and only on the user needs with the time and budget issue keep in mind. XP is highly used to produce a quality product that accepts changing the conditions. According to Williams "XP team members spend few minutes on programming, few minutes on project management, few minutes on design, few minutes on feedback, and few minutes from team-building many times each day (Boehm & Hansen, 2004)."

5) Scrum Model

This process was introduced by two researchers "Jeff Sutherland" and "Ken Schwaber", when working on this for an extended period and at last by the end of June 2006 the first Scrum was professionally implemented, and training on an understanding of this process started (Boehm B. W., 1988). Scrum is an incremental process that provides flexibility to the system and provides help to the team members for the constantly changing environment. Scrum doesn't provide different software methodologies/practices, but it focuses on management practices and development tools to overcome the unintentional complex hindrances during development (Osterweil, 2011). In the scrum, there are some variables in which it works such as customer requirements, time pressure, competition, quality, vision, and recourse. It is an approach that helps development teams to operate independently in the compound environments (Alshamrani & Bahattab, 2015).

6) Feature Driven Development (FDD) Model

In 1997, Jeff De Luca and Peter Coad introduced this process during a very broad and complex project at the United Overseas Bank of Singapore. Jeff De Luca was

the project manager, so he concluded that the task would not complete during the given time by applying available software development methodologies. So, he along with Peter Coad and some others introduced a new process called "Feature Driven Methodology". This model is also called "Modelling in color technique". This approach doesn't focus on the complete development process, but it only emphasizes the design and coding phases. This method has two main tasks: 1) to identify the features to implement, 2) function – to – feature implementation. All the elements are represented using UML diagrams, which is understandable to both developers and the customers if the list of features is as precise as possible it will be beneficial for the developer to maintain the quality and extend the code (Leffingwell, 2010) (Williams & Upchurch, 2001) (Williams, 2003).

2.3 Advantages and Disadvantages

The effects of these software models could not be understood by reading just definitions, but it can be clearer by understanding their advantages and disadvantages regarding different SDLC parameters; also, effects on end-user software products. Table 2 explains some of the major advantages and disadvantages of all six software models discussed above.

Table 2 Advantages and Disadvantages of Software Process Models (Fruhling, McDonald, & Dunbar, 2008) (Jakobsen & Sutherland, 2009)

Advantages		**Disadvantages**	
Traditional Methodology	**Agile Methodology**	**Traditional Methodology**	**Agile Methodology**
Waterfall Model	**Extreme Programming (XP) Model**	**Waterfall Model**	**Extreme Programming (XP) Model**
1) Easy to understand and manage due to distinct phases.	1) This model is suitable for small projects as well as where customers are specific, not general.	1) To adjust a scope or requirements during the development is hazardous.	1) Difficult to manage for large projects where comprehensive documentation is involved.
2) Arrangement and testing of tasks done at the end.	2) It focuses on team coordination.	2) No complete product produced until the end of all the stages.	2) There is no guidance to gather/collect the data.
3) Phases complete at a single time, due to well-documented stages.	3) It emphasizes the final product.	3) Poor model for the compound projects where the rate of change for the requirements is quite moderate.	3) Need experience and skills to handle the XP practices.
Unified Model	**Scrum Model**	**Unified Model**	**Scrum Model**
1) The iterative procedures increase the efficiency of this process.	1) This model provides open discussions in which every team member knows very well his role.	1) Not applicable for small industries.	1) The teams are only responsible for decision making.
2) Testing was done during the iteration and the cost of testing inevitably reduces.	2) Focus on team spirit and communication.	2) If there are no expert project managers, this process is too difficult to apply correctly, complicated too.	2) If any of the team members leave during the project it profoundly affects the development.

3) It works well for small as well as moderate size projects.	3) Frequent meetings and gatherings for better feedback from the customers and stakeholders.	3) In the case of using new technology, the reuse of components will be an issue.	3) The presence of not properly committed team members can cause the project to fail.
Spiral Model	**Feature Driven Development (FDD) Model**	**Spiral Model**	**Feature Driven Development (FDD) Model**
1) Focus on planning and verification in early stages of development.	1) The top priority is to satisfy the customer by providing the early and valuable product.	1) It is not suitable for smaller projects.	1) It depends only on inspections of design, code for quality purposes.
2) Each deliverable must be testable.	2) Teams are highly communicative, but there is a small size of groups to avoid overhead.	2) This phase entirely depends upon risk analysis, that's why it demands higher expertise.	2) It doesn't support refactoring.
3) Works well for those projects where risk analysis is the main problem to resolve.	3) Parking lot charts and feature maps help to track the progress quickly.	3) Hard to handle changing requirements.	3) There is no written documentation for use in the future.

3. DISCUSSION

This section discusses the comparison of traditional and agile development methodologies based on list of key differences, issues, methodology criteria, limitations and cost estimations.

3.1 Comparison Based on Key Differences

Table 3, explains the comparison of traditional and agile development methodology based on major key differences identified from the previous studies.

Table 3 Comparison Based on Key Differences
(Leffingwell, 2010) (Jakobsen & Sutherland, 2009)

Key Difference	Traditional Methodology	Agile Methodology
Customer	Less knowledgeable, co-operative	Dedicated, knowledgeable, representative
Developers	Sufficient skills, plan-determined	Knowledgeable, co-operative, collocated
Objectives	High assurance	Rapid value
Requirements	Stable	Unknown, frequent changes
Size	Larger teams and products	Smaller teams and products
Refactoring	Costly	Cheaper
Risk	Well known, minor effects	Unknown, major effects

3.2 Comparison Based on Issues

Table 4, explains the comparison of traditional and agile development methodology based on major issues identified from the previous studies.

Table 4 Comparison Based on Issues
(Leffingwell, 2010) (Jakobsen & Sutherland, 2009)

Issue	Traditional Methodology	Agile Methodology
Development cycle	Incremental	Linear
Requirements	Clearly defined	Not defined
Documentation	Detailed / heavy	Light
Team members	Distributed teams	Co-location of teams
Development style	Predictive	Adaptive
Client involvement	Low	Active
Project Size	Large	Small
Domain	Predictable	Unpredictable
Team size	Large	Small
Return on investment	End of project	Early in the project

3.3 Comparison Based on Methodology Criteria

Table 5, explains the comparison of traditional and agile development methodology based on different methodology criteria identified from the previous studies.

Table 5 Comparison Based on Methodology Criteria (Schwaber & Beedle, Agile software development with Scrum, 2002)

Methodology Criteria	Traditional Methodology	Agile Methodology
Unclear user requirements	Bad	Excellent
Unfamiliar technology	Good	Bad
Complex systems	Good	Bad
Reliable	Good	Good
Frequent changing	Bad	Excellent
High risk	Good	Bad
Cost	Bad	Excellent

3.4 Limitations of Both Traditional and Agile Methodology

1) Traditional Methodology

The first flaw in this method is the adoption of frequent changes during development. There are two processes “Empirical” and “Defined”, this approach uses a defined process during development. In this process, all of the requirements from the customers described clearly and cost, time is predicted, implemented and results produced. But here is a point to think that if during implementation customer's demands for change in the requirements then there will be an issue because this process doesn't accept frequent changes during development or some predicted variables such as cost and time results in the wrong then there will be overrun of both these.

As far as engineering or large projects are concerned this methodology succeeded (Jensen & Zilmer, 2003). Furthermore, the "Standish Group of Companies" had done a study research survey in which 365 sources and 8380 applications were involved. The sources include IT Executives, Large, medium and small companies. The applications include three categories for the type of projects, category-1: Succeeded Projects, category-2: Failed Projects, category-3: Challenged Projects. The study results that **16.2%** of the projects succeeded on time with mentioned budget and functionalities, **31.1%** of the projects stopped at some point during the development and **52.7%** of the projects challenged due to overrun of budget and time with less mentioned functionalities before development.

This study further provides information about the variables which caused all of these results (Schwaber, Scrum development process, 1997). For successful projects, there was high availability of these three things: User Involvement, Executive Management Support, Clear Statement of Requirements. For failed projects, there was high availability of these three things: Lack of User Input, Incomplete Requirements, and Specifications Changing Requirements and Specifications. For challenging projects, there was high availability of these three things: Incomplete Requirements, Lack of User Involvement, Lack of Resources. (Schwaber, Scrum development process, 1997)

The second one limitation for traditional methodology is managing complexity. The tradition that first plans everything and then implement works well for less complex or small projects but as far as large and complex systems are considered this tradition fails. The solution to managing complexity is only "Simplicity in everything in the system". Here the simplicity means that the team should remove the waste and inventory of the project such as lengthy documentation. Research studies have proven that 25% of the maintenance cost is due to complexity. It is better to keep the rules and everything simple and clear because simple code can be modified easily. To clarify this more, there is another research study done by the "Standish Group of companies". This study states that **45%** of the features and functions that were defined in the large, complex documentations were not implemented in the system, that's why to keep the documentation, and coding simple is the only reason to avoid this. (Schwaber, Scrum development process, 1997)

The third one limitation for traditional methodology is "How this method treats people in developing?" In the traditional method, the people were dealt with as processes; the roles are being assigned to the individuals and assume that they will complete it without inquiring the knowledge that the role suits the person or not? A developer or programmer or any person cannot perform a role perfectly if it is not of his skills or talent. So, the solution to this limitation is that the people must be assigned such works which can they do with interest. Also, they must be appreciated by the management.

2) Agile Methodology

The first limitation of the agile methodology is that it is not suitable for government agencies, large organizations such as banks, insurance companies, etc. or long-term maintenance of the systems because these both involve detailed and large documentations that were highly ignored in this methodology. So, these types of organizations and the systems are satisfied with the traditional method because their primary requirement is fulfilled there (Anderson, 2004). In agility, the work of documentation is shifted towards the people or team members because it is an

assumption of this methodology is that all the team members will be there until the end of the project. But in most of the cases, for large systems, this doesn't happen because due to the long-term deadline, anything can happen to the team members and the team can be disturbed. Moreover, documentation is necessary for maintenance, usage of the system for a long span of time.

The second limitation is that the agile methodology heavily depends upon the involvement of the user or the people of the organization. So, the success of the project is only dependent on the communication and performance of the people factor of the team. If there is the best process implemented, but there is no best staff, then this methodology fails. Also, if the level of the developers is a beginner or there is a communication gap between the developer and the customer this method fails. The only success criteria for the people are that they must be skilled and talented. (Anderson, 2004). In support to this limitation, Boehm contends noting that, "A significant consideration here is the unavoidable statistic that 49.9999 % of the world's software developers are below average" (Khramtchenko, 2004). The agile methods try to have a cream of skilled people to work because the agile wants the people to understand or tackle those jobs which were tackled by documentation in the traditional methodologies.

The discussion of the people factor leads to another thing that by having capable and skilled people, there is no need for best practices to work if the people are best enough then they can collaborate with any practice. Another side of the people factor is the involvement of the customer. But what happens when there are multiple clients, conflicts of the requirements or the customers are not applicable to providing needs then at this point the traditional methodology works best due to documentation, reviews, and planning.

The third limitation is that how it works with larger teams, probably the most significant limitation because for small teams, it works best, but for large, there is a lot of issues to consider (Anderson, 2004). For team size greater than 20, it becomes difficult for agile to manage the face-to-face conversation and the setup becomes more complicated for the developer.

3.5 Cost Estimation in Both Traditional and Agile Methodology

The cost estimation process for the software begins in the planning phase of the SDLC (Software Development Life Cycle). When the project manager is assigned a project, first he thinks of what resources will be needed? I-e: hardware, software, testing tools, employees, etc. After the planning of the tasks and identification of the resources is finished the estimation process starts from the listed needed resources. One important point to discuss is, if the project manager identifies the wrong resources, so all the estimates will be a mistake and the project will be over budget. To overcome this issue, the project manager must use some standard cost estimation techniques to calculate all the estimations. Table 6, focuses on success factors for cost estimation in both of the methodologies, which is included at the time of budget allocation then the project will never over planned and found to be key factors in this.

Table 6 Success Factors for Cost Estimation (Goyal, 2008)

Traditional Methodology	Agile Methodology
Entertainment Cost - The entertainment cost is a severe reason for over budget of	**Active Customer Involvement -** Agile processes highly support active

a software project. Entertainment cost doesn't include just client-side costs, but it also includes all of the extra expenses that project manager does during the project, such as outside meeting with higher management, suppliers, stakeholders, etc. Usually, the project manager doesn't include this cost at the time budget allocation for the project. They decide to add this cost to the project later. At that point project allocated budget starts overrun and exceeded the mentioned budget. So, the conclusion is that the project manager must include entertainment cost at the date of budget allocation.	participation of the clients in the project. According to agility the active participation of the customer helps to have a clear and concise picture of the whole system and the expected end product. The active participation of the client also helps the developer to get the objectives and requirements from the customers when a customer identifies the needs. Also, when the end product is ready, the client can verify that if it is according to his/her needs or not. This approach highly reduces the cost of the review of the system again and again.
Sponsor's Role - A sponsor is a person who is responsible for allocation of resources and budget for the project. He is an indirect person involved in the project, because the project manager must have to continuously report to the sponsor for the allocation of the resources, also to inform him about the budget that whether his budget is utilized or not. Therefore, the involvement of the sponsor is crucial for proper identification of the resources because this will help to estimate the actual costs. If the only team members assume resources needed for the project, then this will inevitably cause the estimation to be high for not resource requirements.	**Strong Communication** - Communication is the task of conveying information between two people or group of individuals. The purpose of communication is to discuss something at any place. Most of the researchers suggested that communication has a vital role in software development. In software development, communication is between the customer and the management. Active communication between both of these results in a successful software product. Daily meeting and talks between both of these clarify the requirements and scope of the product. More communication will make their relationship strong and thus resulting in a successful software product. Also, a review of requirements and short-term outputs in the meetings will inevitably reduce the cost of the system.
Suitable Estimation Technique - In software development process, there are some estimation techniques used for estimation of the expenses. For example, a top-down approach, price-to-win, expert judgment, bottom-up approach, rules of thumb, etc. From the past few years, researchers are involved in creating such a technique that can provide accurate results for every scenario. But still, they are failed, because of changing requirements and	**Simplicity -** Agile offers simplicity in its projects because it helps the project team to complete the project in a shorter time as the process is not so much complex. Also, simplicity clarifies that which resources are needed, or which features needs to add into the design and the code. This approach reduces the time and cost estimation up to the individual level because there will be no wastage of not significant components and resources. Simplicity has got three main points:

other factors in a project. Choosing the right estimation technique is critical to generating accurate estimates for the project. Therefore till today, the best solution is to pick a method according to your project and circumstances. Also, it is not possible that applying more than one method will produce more than one result, but those results should be accurate.	1) do less, 2) do better and 3) do swarm. Do fewer means that there should be fewer tasks, fewer documents, and less managerial reports. Do better means it has its specific task in the design phase. Do swarms means it simplify the complexity generated during the development.

4. CONCLUSION

During reviewing and studying, one thing is very clear that selection of the method is only dependent on the type of project, resources needed, estimations, etc. While discussing and studying traditional methods, some of the major points that concluded were this method is best suitable for the compound as well as long term (up to some years) projects, because the main feature supporting this is the documentation. Another thing appears that this methodology highly resists change, whether it is the change of requirements, resources or anything because at the early phases of this method the requirements and other resources are being fixed for the whole project and according to them the work is started. Basically, agile methodologies are being adopted for the disadvantages by the traditional methods in the projects like small, business, frequent changeable, short term, minimal cost, etc. So, in the agile instead of sequential approach, the processes are break down into the small phases. Short outcomes after each step shown for the client to get feedback and if there is any change in the demand that needs to change at that time of the comments. If the estimates are very accurate and employees are working timely then surely a quality product will be generated quickly. So, the most important thing to conclude in last is that today the environment is changing very frequently, so acceptance of the agile over traditional methodology will surely help most of the business organizations to generate quality products. But the importance of traditional method also cannot be denied because it has its functional areas where it can perform better than agile. So, at last the conclusion is that it depends on the type of project that we are going to build because no single solution can solve all the general problems.

REFERENCES

Aitken, A., & Ilango, V. (2013). A comparative analysis of traditional software engineering and agile software development. *2013 46th Hawaii International Conference on System Sciences*, (pp. 4751-4760).

Alshamrani, A., & Bahattab, A. (2015). A comparison between three SDLC models waterfall model, spiral model, and Incremental/Iterative model. *International Journal of Computer Science Issues (IJCSI), 12*, 106.

Anderson, D. (2004). Feature-Driven Development: towards a TOC, Lean and Six Sigma solution for software engineering, Theory of Constraints. *International Certification Organization, Microsoft*.

Awad, M. A. (2005). A comparison between agile and traditional software development methodologies. *University of Western Australia*, 30.

Boehm, B. W. (1988). A spiral model of software development and enhancement. *Computer*, 61-72.

Boehm, B., & Hansen, W. J. (2004). Spiral development: Experience, principles and refinements, 2000. *DTIC Document*.

Fruhling, A., McDonald, P., & Dunbar, C. (2008). A case study: introducing extreme programming in a US government system development project. *Proceedings of the 41st Annual Hawaii International Conference on System Sciences (HICSS 2008)*, (pp. 464-464).

Goyal, S. (2008). Major seminar on feature driven development. *Jennifer Schiller Chair of Applied Software Engineering*.

Jakobsen, C. R., & Sutherland, J. (2009). Scrum and CMMI going from good to great. *2009 Agile Conference*, (pp. 333-337).

Jensen, B., & Zilmer, A. (2003). Cross-continent development using Scrum and XP. *International Conference on Extreme Programming and Agile Processes in Software Engineering*, (pp. 146-153).

Khramtchenko, S. (2004). Comparing eXtreme Programming and Feature Driven Development in academic and regulated environments. *Feature Driven Development*.

Kroll, P., & Kruchten, P. (2003). *The rational unified process made easy: a practitioner's guide to the RUP*. Addison-Wesley Professional.

Leffingwell, D. (2010). *Agile software requirements: lean requirements practices for teams, programs, and the enterprise.* Addison-Wesley Professional.

Munassar, N. M., & Govardhan, A. (2010). A comparison between five models of software engineering. *International Journal of Computer Science Issues (IJCSI), 7*, 94.

Osterweil, L. J. (2011). A Process Programmer Looks at the Spiral Model: A Tribute to the Deep Insights of Barry W. Boehm. *Int. J. Software and Informatics, 5*, 457-474.

Process, R. U. (2001). Best practices for software development teams. *A Rational Software Corporation White Paper. TP026B, Rev, 11*.

Schwaber, K. (1997). Scrum development process. In *Business object design and implementation* (pp. 117-134). Springer.

Schwaber, K., & Beedle, M. (2002). *Agile software development with Scrum* (Vol. 1). Prentice Hall Upper Saddle River.

Williams, L. (2003). The xp programmer: the few-minutes programmer. *IEEE software*, *20*, 16.

Williams, L., & Cockburn, A. (2003). Agile software development: it's about feedback and change. *IEEE Computer*, *36*, 39-43.

Williams, L., & Upchurch, R. (2001). Extreme programming for software engineering education? *31st Annual Frontiers in Education Conference. Impact on Engineering and Science Education. Conference Proceedings (Cat. No. 01CH37193)*, *1*, pp. T2D--12.