International Journal of Advance Research in Science and Engineering
Volume No.07, Special Issue No.03, February 2018
www.ijarse.com
IJARSE
ISSN: 2319-8354

TRADITIONAL APPROACH TO AGILE APPROACH IN SOFTWARE DEVELOPMENT

Sampada Chaudhari[1],Prateeksha Chouksey[2],
Priyanka Lonkar[3]

[1,2,3]Dept. of Computer Engg., BSCOER, Pune , (India)

ABSTRACT

Selecting the right approach for software development has many factors like the project quality, costs and schedule. Traditional approaches suffice when projects are based on rigidly followed processes to ensure quality output. But such methods do not encompass or cannot changes until the whole cycle is complete. To be able to sustain in such an environment of constant transformation, a business needs to be adaptive. The key solution is to adopt agile methodologies for business architecture and information technology. With such an approach, organizations can quickly respond to customer demands and efficiently adapt to market changes. It dramatically increases flexibility and decreases development time. This paper mainly focuses on the comparison between traditional approach and agile approach in terms of development, management and adaptation of new technologies along with its effects.

I.INTRODUCTION

The system development life cycle (SDLC) is a conceptual model used to develop information systems with well-defined phases. The phases are: user requirements definition, system, requirements definition, analysis and system design, system development, testing, implementation and maintenance. Some of the models use rare Waterfall, Spiral and Rapid Prototyping. These models are called plan-driven, lying on end of the "planning emphasis". At the other end of this are Agile methods, which are considered adaptive rather than predictive. A software system is built in such a way that it can perform complex tasks and computations on the behest of the user. The process of building software requires a rigorous attention to detail and a general guiding algorithm. The traditional and agile processes essentially are manifestos for software development ideologies, more than anything else. They can be further subdivided into more specific processes that form the basic plan of each unique software development life cycle. The agile methodology post-dates the traditional one in the evolution of the software development processes.

1.1 Traditional Software Development

Traditional methodologies are characterized by a sequential series of steps like requirement definition, planning, building, testing and deployment. First, the client requirements are carefully documented to the fullest extent. Then, the general architecture of the software is visualized and the actual coding commences. Then comes the various types of testing and the final deployment. The basic idea here is the detailed visualization of the finished

project before the building starts, and working one's way through to the visualized finished structure.Traditional methodologies are plan driven in which work begins with the elicitation and documentation of a complete set of requirements, followed by architectural and high level design development and inspection. Due to these heavy aspects, this methodology became to be known as heavyweight. Some practitioners found this process centric view to software development frustrating and pose difficulties when change rates are still relatively low.Traditionally, the process of software development included rigid software development models. Processes included a structured layout of a step by step approach from requirements gathering to final testing and release of the product. Well, the processes were mostly aimed towards creating a flawless product in accordance to software quality standards.There are two major ways to accomplish a successful project – SDLC (Software Development Life Cycle) and PDLC (Project Development Life Cycle). PDLC is a set of activities that go hand in hand and control the projects.

1.2. Agile Software Development

As its name suggests, the agile method of developing software is a lot less rigorous than the former. The crux here is incremental and iterative development, where phases of the process are revisited time and again. Agile developers recognize that software is not a large block of a structure, but an incredibly organic entity with complex moving parts interacting with each other. Thus, they give more importance to adaptability and constant compatibility testing.

II.LITERATURE SURVEY

Several Software Development Lifecycle Models (SDLCs) are in existence. Over the time different people have proposed different models to meet the industrial demands. Any SDLC process model should be a repeatable, clearly documented, highly-effective and must be based on the best industrial practices. The traditional SDLC process models provide very insightful theory and helpful best practices, but do not provide the practical details for daily application. SDLC pro-cess models are rarely used by organizations for the purpose they are designed and developed for. Another primary reason for not using these models is due to lack of their suitability for real life projects - which led to software crisis [1]

We lack well defined characteristic parameters for any SDLC model. Without applying the process model in real project, we do not have adequate metric to analyze the suit-ability and goodness of such models. As requirements are changed frequently, there is a need of streamlined flexible approach to manage these requirement changes within the SDLC model. To develop quality software on a predictable schedule, the re-quirements must be established and maintained with reasonable stability throughout the development cycle. Changes will have to be made, but they must be managed and introduced in an orderly way. Hence, change management is a critical part of any SDLC model [3].

If large numbers of people are involved and scattered over different development centres, the process model must provide mechanism for better coordination among the project stakeholders. Additional costs and overheads

are the primary barrier in process model implementation. For this reasons, many organizations do not implement or follow any process model.

In most of the common process model, there is no direct communication among customer, development team or project management team throutout the development process. In traditional models, management plays the vital role. As a result always there remains some communication gap and some missing or hidden information yet to convey to the development team but with the management. As a result, often proper requirements remain unspoken or hidden to the development team. Even conveyance of information might cause a loss of knowledge, as great amount of data remains with its carrier and never get handed off to others. The lack of direct contact between the development team and the customers could encumber the process of specifying requirements for the future. In turn, handoffs among functions can cause delays and increasing risks of information being misunderstood. Level of details is varying depending on representatives between customer and developer. As a result, the developed system is frequently not satisfactory or even lead to project failure [6].

The process model must focus on identifying the errors in the same or closest phase of the SDLC process to avoid or reduce the redo-work and cost. Most of the existing traditional SDLC process models don't involve management team directly with the development team. Hence, the project management team does not have direct communication with the development and associated members. The management just remains as a silent intermediate communication body. Thus, proper management observation and control is hidden in the development process. As a result, the development process lacks proper management supervision and controls. In addition, the project has to suffer from resource shortage, risk handling, coordination and many other conflicts and problems [5].

Testing phase is the highest risky phase. Thus, all problems, bugs, and risks are discovered too late when the recovering from these problems. It requires large rework which consumes time, cost, and effort. A potential source of risk resides in the relatively long stages of any process model, which makes it difficult to estimate, time, cost, and other resources required to complete each stage successfully.

III.CHARACTERISTICS OF TRADITIONAL AND AGILE METHODOLOGIES

1.1 Traditional Methodologies Characteristics

Traditional methodologies have been around for a very long time. They impose a disciplined process upon software development with the aim of making software development more predictable and more efficient. They have not been noted to be very successful and are even less noted for being popular.

1.1.1 Predictive approach – Traditional methodologies have a tendency to first plan out a large part of the software process in great detail for a long span of time. This approach follows an engineering discipline where the development is predictive and repeatable. A lot of emphasis is put on the drawings focusing on the need of the system and how to resolve those needs efficiently. The drawings are then handed over to another group who are responsible for building the system. It is predicted that the building process will

follow the drawings. The drawings specify how they need to build the system; it acts as the foundation to the construction process. As well, the plan predicts the task delegation for the construction team and reasonably predicts the schedule and budget for construction.

1.1.2 Comprehensive Documentation – Traditional software development view the requirements document as the key piece of documentation. A main process in traditional methodologies is the big design upfront (BDUF) process, in which a belief that it is possible to gather all of a customer's requirements, upfront, prior to writing any code. Again this approach is a success in engineering disciplines which makes it attractive to the software industry. To gather all the requirements, get a sign off from the customer and then order the procedures (more documentation) to limit and control all changes does give the project a limit of predictability. Predictability is very important in software projects that are life critical.

1.1.3 Process Oriented - The goal of traditional methodologies is to define a process that will work well for whoever happens to be using it. The process would consist of certain tasks that must be performed by the managers, designers, coders, testers etc. For each of these tasks there is a well-defined procedure.

1.1.4 Tool Oriented – Project management tools, Code editors, compilers, etc. must be in use for completion and delivery of each task.

1.2 Agile Methodologies Characteristics

The following principles of agile methodologies are seen as the main differences between agile and traditional:

1.2.1 People Oriented- Agile methodologies consider people – customers, developers, stakeholders, and end users – as the most important factor of software methodologies. If the people on the project are good enough, they can use almost any process and accomplish their assignment. If they are not good enough, no process will repair their inadequacy.

1.2.2 Adaptive – The participants in an agile process are not afraid of change. Agilists welcome changes at all stages of the project. They view changes to the requirements as good things, because they mean that the team has learned more about what it will take to satisfy the market. Today the challenge is not stopping change but rather determining how to better handle changes that occur throughout a project.

1.2.3 Conformance to Actual – Agile methodologies value conformance to the actual results as opposed to conformance to the detailed plan. Each iteration or development cycle adds business value to the ongoing product. For agilists, the decision on whether business value has been added or not is not given by the developers but instead by end users and customers.

1.2.4 Balancing Flexibility and Planning – Plans are important, but the problem is that software projects cannot be accurately predicted far into the future, because there are so many variables to take into account. In this view one of the main sources of complexity is the irreversibility of decisions. The consequence for agile design is that designers need to think about how they can avoid irreversibility in their decisions.

1.2.5 **Empirical Process –** Agile methods develop software as an empirical (or nonlinear) process. In software development it cannot be considered a defined process because too much change occurs during the time that the team is developing the product.

1.2.6 **Decentralized Approach –** Integrating a decentralized management style can severely impact a software project because it could save a lot of time than an autocratic management process. Agile software development spreads out the decision making to the developers. This does not mean that the developers take on the role of management. Management is still needed to remove roadblocks standing in the way of progress. However management recognizes the expertise of the technical team to make technical decisions without their permission.

1.2.7 **Simplicity –** Agile teams always take the simplest path that is consistent with their goals. Never produce more than what is necessary and never produce documents attempting to predict the future as documents will become outdated.

1.2.8 **Collaboration –** Agile methods involve customer feedback on a regular and frequent basis. The customer of the software works closely with the development team, providing frequent feedback on their efforts. As well, constant collaboration between agile team members is essential. Due to the decentralized approach of the agile methods, collaboration encourages discussion.

1.2.9 **Small Self-organizing teams –** An agile team is a self-organizing team. Responsibilities are communicated to the team as a whole, and the team determines the best way to fulfill them. Agile teams discuss and communicate together on all aspects of the project. That is why agility works well in small teams.

III.DIFFERENCES BETWEEN TRADITIONAL AND AGILE DEVELOPMENT

Table- Difference between Traditional and Agile Aproach

	Traditional development	**Agile development**
Fundamental hypothesis	Systems are fully specifiable, predictable and are developed through extended and detailed planning	High quality adaptive software is developed by small teams that use the principle of continuous improvement of design and testing based on fast feed-back and change
Management style	Command and control	Leadership and collaboration
Knowledge management	Explicit	Tacit
Communication	Formal	Informal

Development model	Life cycle model (waterfall, spiral or modified models)	Evolutionary-delivery model
Organizational structure	Mechanic (bureaucratic, high formalization), targeting large organization	Organic (flexible and participative, encourages social cooperation), targeting small and medium organizations
Quality control	Difficult planning and strict control. Difficult and late testing	Permanent control or requirements, design and solutions. Permanent testing
User requirements	Detailed and defined before coding/implementation	Interactive input
Development direction	Fixed	Easily changeable
Testing	After coding is completed	Every iteration
Client involvement	Low	High
Developers	Oriented on plan, with adequate abilities, access to external knowledge	Agile, with advanced knowledge, co-located and cooperative
Requirements	Very stable, known in advance	Emergent, with rapid changes
Remodeling	Expensive	Not expensive
Size	Large teams and projects	Small teams and projects
Primary objectives	High safety	Quick value

IV.CONCLUSION

No matter what model is chosen for developing software applications, this activity involves complex processes that are often predisposed to errors. That is why, beyond agility or traditionalism, an important role goes to testing and validation. Any high quality software system, with professional development and implementation must be tested and validated before going into production. The client must know that the system was developed and implemented according to the project specifications. Also, the client must be sure the project functionality is correct.

REFERENCES

[1] ArdhenduMandal and S. C. Pal, *" Investigating and Analysing the Desired Characteristics of Software Development Lifecycle (SDLC) Models", INTERNATIONAL JOURNAL OF SOFTWARE ENGINEERING RESEARCH & PRACTICES VOL.2, ISSUE 4, OTOBER, 2012.*

[2] http://www.techopedia.com/definition/22193/software-development-life-cycle-sdlc

[3] Martin Michlmayr, Francis Hunt, David Probert, "Quality Practices and Problems in Free Software Projects", Proceedings of the First International Conference on Open Source Systems Genova, 11th-15th July 2005

[4] http://www.tutorialspoint.com/sdlc/sdlc_overview.htm

[5] Barry Boehm, Richard Turner, "Management Challenges to Implementing Agile Processes in Traditional Development Organizations", Published by the IEEE Computer Society, 0740-7459/05/$20.00 © 2005 IEEE.

[6] Andrei Antanovich, Anastasia Sheyko& Brian Katumba, "Bottlenecks in the Development Life Cycle of a Feature, Bachelor of Science in Software Engineering and Management", Thesis Report No. 2010:012 ISSN: 1651-4769.

[7] http://istqbexamcertification.com/whatare-the-software-development-models/

[8] http://www.slideshare.net/J.T.A.JONES/software-development-life-cycle-model-1392777

[9] http://www.tutorialspoint.com/sdlc/sdlc_overv iew.htm

[10] S. Nerur, R. Mahapatra, G. Mangalaraj, Challenges of migrating to agile methodologies, Communications of the ACM (May) (2005) 72– 78.

[11] Get Ready for Agile Methods, with Care,BarryBohem, IEEE Ianuarie 2002, tabel 1, pg. 68

[12] http://www.producao.ufrgs.br/arquivos/disciplinas/507_artigo_3_empirical_systematic_revew.pdf