DEVELOPMENT PROCESS FOR CONTROLLABLE SOFTWARE

Eze Nicholas Ude
+2348078471262, +2347036850857 dr.nickeze@yahoo.in

Department of Science and Computer Education,
Enugu State University of Science and Technology,
Enugu State Nigeria.

Abstract
A major challenge in any software engineering endeavor is taking a poorly organized, uncertain, inconsistent, incomplete, and over generalized requirements specification and turning it into a well-structured design. In order to manage quality and to save cost and time, many companies are shifting from traditional software development lifecycle models to the agile environment. With the emergence of the Internet, software development has become an integral part of almost every facet of business today. Because consumers have a surmounting demand for immediacy and convenience, companies are pressured to add web-based services to their product offerings. Therefore, an increasing number of resources are being allocated to the development of profitable controllable software to meet customer needs. Because companies desire to maximize their profits, an efficient allocation of these resources is necessary to minimize costs. This can be achieved by implementing a process model that best converts their resources to quality products. Agile software development is a relatively new framework aimed at reducing risk and production costs. It is based on iterative development and continuous feedback from all stakeholders throughout the development cycle. The switch to an agile process model from a traditional waterfall process model can reduce the risk associated with producing a large-scale software application by decreasing lead times and increasing team morale and productivity. My literature review and initial findings suggest that firms across industries can benefit from incorporating some degree of agility in their development process. This paper therefore aims to describe the impact and comparison of various traditional methodologies and new methodology. This paper explores the reasons for which software industries shifted from Traditional software development to Agile environment. The researcher has performed comparative studies of different software development methodologies, when and where they can be used. This paper also gives introduction to agile software development methodology and how to apply them.

Keywords *— Software Development Life Cycle, Agile Development Environment, Agile Software Development Methods, Requirement Engineering, Traditional Models. Software Development.*

1. Introduction

To maintain the software expansion process, many effective quality systems are developed; which address an organization's commercial requirements. Designers employ various types of system development process model to direct the project's life cycle. Various activities may be done in various phases of software development process by specific individual or team. Classic activities performed in software development process includes: System planning, System requirements and benefits exploration, Project Endorsement and Project Scoping, System Design and Development, Software Integration & Testing, System Integration & Testing and Documentation, Implementation and Maintenance. The major issue in software development is to regulate, how to instrument, using certain skills and within certain boundaries. Software development lifecycle (SDLC) gives a brief outlook of how the dissimilar problem-solving events may be done in various chapters by an discrete or team doing software advance. The various SDLCs models are- 1) Waterfall 2) Iterative 3) Iterative and Incremental 4) Evolutionary Prototyping 5) Ad-hoc or Code-And-Fix SDLC. Numerous agile ideas have been around since 70's or smooth before and their substance is documented as a response against various traditional methods. Agile methods on an entire are new; they have healthy roots in the history of software engineering. The term agile can be defined by quickness, correctness, and ease of quantity and it has also increased the public consideration in late 1990's. Agile approaches were well recognized to progress systems more quickly with partial time spent on analysis and design. Agile approaches to the development process for controllable software model is necessary to define a timeline of deliverables for the software project and to ensure that every member of the development team, from management to engineers to consumers, understands the expectations of the workflow. While a waterfall model has been used for many years, companies are beginning to realize its inherent restrictions. Due to its linear timeline of creating requirements, and implementing the complete functionality, it does not adapt well to changing consumer needs and thus exposes the project to substantial risk. My research explores the impact of switching to an agile methodology on such risk and associated costs. Agile employs an iterative development process where functionality is implemented in two-to four-week iterations. Therefore, all three main components of software development are repeated throughout the development cycle. This allows for

change in consumer desires, as a small amount of functionality is produced every few weeks and may be easily altered. Risk is therefore less than that under the waterfall model, which would then reduce the cost of capital for software projects under the agile framework. Because this process is very adaptive and open to continuous feedback, the final product is almost perfectly aligned with customer expectations, thus resulting in high satisfaction and demand, which yield high profits. Only a sufficient implementation of an agile method, that is, a sufficient degree of agility, however, can lead to such reduced software development costs. My research investigates the advantages and disadvantages of both process models to determine the impact of a switch in a business context. More specifically, I begin with an analysis of the general expert consensus on the limitations of both traditional and agile methods and the benefits of an agile process model. Next, I explore the effect of switching from a traditional, plan-driven model to agile methods on several measures of cost, quality, and customer satisfaction. Finally, I analyze the degree of agility, defined as the fit between the actual practices of a firm and the espoused agile practices, employed by most companies and its correlation with reduced risk and costs.

1.1 Background

Various lifecycle development models have been created and also applied to large-scale system and software development projects, which are used in government sector various industries and many more different areas. As a solution for the traditional software expansion approaches agile software development approaches (ASDMs) were well-known. Overhead of the historical four centuries, outmoded waterfall expansion methods have been broadly rummage-sale for significant projects in the software manufacturing and in the government sector because of their straightforward, logical, and planned nature and for their capability to provide inevitability, control, and high declaration. According to survey, the waterfall model is being used by a huge portion of the software engineering world, Subhod (2012). In another development in a survey of 1,027 IT projects in United Kingdom, Thomas (2001) have reported about scope management related to attempting waterfall practices and clarify that it was the solitary largest donating factor for dissatisfaction. Laffingwell (2007) has defined four main key expectations with the waterfall model which turned out to be not precise. The practices of prototyping and the object-oriented models are the main conducts to contrivance the waterfall, incremental, and spiral models. Agile method is not a set of tools neither a single methodology, but according to the philosophy which was formulated in 2001 with an initial 17 participants, it was in adjustment to the document-driven software development practices - ex waterfall. Hue (2013) designates; that in agile system enterprise is used in place of formal construction, which can present a story of how system is worked by unfolding all segments and adornment of the system. There have been few attempts at measuring and proving the impact of switching to an agile development process model on costs and overall software quality. Jaana Nyfjord and Mira Kajko-Mattsson (2013), from Stockholm University are in the process of integrating agile methods with risk management, while Kai Petersen and Claes Wohlin discovered various improvements by moving from a plan-driven to an incremental software development approach by performing a case study at Ericsson AB in Sweden Kai Petersen and Claes Wohlin (2012). However, there have been even fewer attempts to quantize the effect of such a switch.

2. Literature Review

Software development is an area or department where we develop a software product. It is a computer programming where we use coding. For developing any new software, we analyses all things like research, prototyping, modification, reuse, maintenance, or any more other activities. We also care about the cost of the software, efficiency, reliability, fits according to requirements and more basic points. Yourdon, (2008) defines Software development as the computer programming, documenting, testing, and bug fixing involved in creating and maintaining applications and frameworks involved in a software release life cycle and resulting in a software product. Goldberg, (2006), defines the term software development as a process of writing and maintaining the source code, but in a broader sense of the term it includes all that is involved between the conception of the desired software through to the final manifestation of the software, ideally in a planned and structured process. Therefore, software development may include research, new development, prototyping, modification, reuse, re-engineering, maintenance, or any other activities that result in software products.

Software development in an amateur perspective can be understood as transplanting and translating people's idea into computer to gain certain functionality and to achieve some goal, usually under a team of many different roles, the developer, usually refer to the coder or programmer, using programming language to realize the intended function in software, or using model or workflow in recent years which allow coder to program in a visualized style, there are other roles, like analysts, architects, sometime designer, and program managers, testers so on. The coder or developer's job is mainly to follow the architecture designed by someone else, so these two concepts are dramatically different. There are many phases and models to develop software: they are Waterfall model, Spiral model, Iterative and incremental development, Agile development, Rapid application development, Code and fix models. However, a survey was conducted and in that according to Dennis, Wixom, and Teagarden (2005), the approximating phase, in waterfall model occupies typically about 15% of the total Systems of Development Life Cycle and in analysis phase, about 15% of the model, analyses the current system about its problems and then also classifies how to design the new system through requirements assembling. Now, coming to the agile methodologies, a survey conducted in 2008 shows that 67% of agile practitioners experienced improvements in their software development process. And also a survey conducted by Dr. Dobb's shows that 41% of development projects have now adopted agile methodology, and also agile techniques are being used on 65% of such projects. A research study conducted by the Standish Group regarding 8,380 projects from 365 respondents, shows that only a small percentage of projects (16.2%) with traditional methods were completed on-time and budget with all features and functions specified and 52.7% of the projects were completed over-budget, and time estimate, and offering less features and functions; 31.1% of projects were cancelled at some point during the development cycle. In a study of failure influences in 1027 IT projects in the UK, scope management associated with Waterfall practices was quoted to be the largest difficulties in 82% of the projects and approximately 13% of the projects surveyed didn't fail (Taylor 2000). A large project study, Chaos 2000 by The Standish Group showed that 45% of requirements in early specifications were never used (Johnson 2002). Parrish (2004) argues that agile software development methods (ASDMs) provide increased quality, shorter time to market, better efficiency, and greater customer satisfaction. Miller and Larson(2005) also believe that ASDMs emphasize close collaboration between the users and developers of a project, and relatively quick development cycles that can react to changing requirements. Australian group, Shine Technologies (2003) surveyed 131 defendants of teams and companies that had applied agile methodologies [1] 93% stated that productivity was better ,49% stated that costs were reduced and [3] 88% stated that quality was much better, and 83% stated that business satisfaction was also great. Spencer (2005) stated that "our implementation of agile applies helps us find bugs earlier, and also to achieve higher quality, and helps us work well with QA.

2.1 Software Development Life Cycles

Software development project has to go through the various stages such as- Requirement gathering, writing useful specifications, Creating policy design documents, Implementation coding testing and also quality assurance.

2.1.1 Requirements Engineering Phase:

Plan-driven, or traditional, methodologies rely on extensive and comprehensive planning before the implementation of any functionality can begin. Because many documented and validated requirements, which were compiled only at the first stage of the development process, must be discarded and reworked throughout the implementation and testing phases, due to changing customer needs, much time and effort is wasted outlining an exhaustive list of requirements. Therefore the fraction of implemented requirements over the total number of requirements written is small, which suggests waste. At the same time, because a plan-driven process model is not adaptive to a changing market, many of the features that are implemented in the final product are not needed or used by customers. Thus requirements become obsolete due to long lead times (the time between project conception and implementation). Such waste increases the cost of development because those resources allocated to writing an excessive number of requirements, over those desired by current customer needs, do not produce any output of value. Those resources could have been allocated to other phases of the development process, or to another project, to obtain a positive return on

investment. In line with this, Stephenson (2009), identified some constraints in the requirement stage of software development as cost and deadline (for example, if the finished product must cost less than $370,000 and must be completed within 14 months), but a variety of other constraints are often present, such as reliability (the product must be operational 99% of the time) or the size of the object code (it has to run on the client's personal computer). A major issue in software development is this so-called moving target problem when the client changes the requirements during development. One reason this occurs is an unforeseeable change in circumstances. For example, if a company expands its operations, or is taken over by another company then many products will have to be modified, including those still under development. However, the major cause of the moving target problem is a client who keeps changing his or her mind.

2.1.2 Specification Phase:

Once the client agrees that the developers understand the requirements, the specification document is drawn up by the specification team. As opposed to the informal requirements phase, the specification document (or specifications) explicitly describes the functionality of the product, that is, precisely what the product is supposed to do, and lists any constraints that the product must satisfy. As already known fact, a major source of faults in delivered software is faults in the specification document that are not detected until the software has gone into operations mode. That is why Schach (2015), in his work maintained that a variety of difficulties can arise during the specification phase. One possible mistake that can be made by the specification team is that the specifications may be ambiguous—certain sentences or sections may have more than one possible valid interpretation. Consider the specification, "a part record and a plant record are read from the database. If it contains the letter a directly followed by the letter q, then compute the cost of transporting that part to that plant." To what does the *it* in the preceding sentence refer: the part record or the plant record? In fact, the *it* could even refer to the database! The specifications may also be incomplete, that is, some relevant fact or requirement may have been omitted. For instance, the specifications may not state what actions are to be taken if the input data contain errors. Moreover, the specification may be contradictory. For example, in one place in the specification document for a product that controls a fermentation process, it is stated that if the pressure exceeds 35 PSI, then valve m 17 must immediately be shut. However, in another place it is stated that if the pressure exceeds 35 PSI, then the operator must immediately be alerted; only if the operator takes no remedial action within 30 seconds should valve ml7 be shut automatically. Software development cannot proceed until such problems in the specification document have been corrected.

2.1.3 Planning phase

No client will authorize a software project without knowing in advance how long the project will take and how much it will cost. From the viewpoint of the developers, these two items are just as important. If the developers underestimate the cost of a project, then the client will pay the agreed fee, which may be significantly less than the actual cost to the developers. Conversely, if the developers overestimate what the project will cost, then the client may turn the project down or have the job done by other developers whose estimate is more reasonable. Similar issues arise with regard to duration estimation. If the developers underestimate how long it will take to complete a project, then the resulting late delivery of the product will, at best, result in a loss of confidence on the pan of the client. At worst, lateness penalty clauses in the contract will be invoked, causing the developers to suffer financially. Again, if the developers overestimate how long it will take for the product to be delivered, the client may well award the job to developers who promise faster delivery.

2.1.4 Design Phase

The specifications of a product spell out what the product is to do. The aim of the design phase is to determine how the product is to do it. Starting with the specification document, the design team determines the internal structure of the product. During the design phase, algorithms are selected and data structures chosen. The inputs to and outputs from the product are laid down in the specifications, as are all other external aspects of the product. However, Shapiro (2014), maintained that, in practice the above assertion is difficult to achieve. Deadline constraints in the real world are such that designers struggle against the clock

to complete a design that will satisfy the original specification document, without worrying about any later enhancements. In addition there is no way of determining, while the product is still at the design phase, what all possible future enhancements might be. And finally, if the design has to take all future possibilities into account, at best it will be unwieldy; at worst it will be so complicated that implementation will be impossible. Design reviews are similar to the reviews that the specifications undergo. This according to Weingberg (2011), in view of the technical nature of most design documents, maintained that the client is not usually present during design phase and so there may arise some faults like logic faults, interface faults, lack of exception handling (processing of error conditions), and, most important, nonconformance to the specifications.

2.1.5 Implementation Phase

During the implementation phase, the various component modules of the design are coded. The major documentation associated with implementation is the source code itself, suitably commented. But Kelly, Sherif and Hops (1994) stated that the major limitation associated with this particular phase is not being able to detect programming faults. This is so in that changing requirement necessitated by unclear and unambiguous goals and objectives will have to be fed back into the design phase and latter into the coding/testing and system integration phases, thus creating a ripple effect with repercussions for the other phases. This is too shouting during software implementation.

2.1.6 Integration Phase

The next stage is to combine the modules and determine whether the product as a whole functions correctly. The way in which the modules are integrated (all at once or one at a time) and the specific order (from top to bottom in the module interconnection diagram or bottom to top) can have a critical influence on the quality of the resulting product. For example, suppose the product is integrated bottom-up. If there is a major design fault, then it will show up late, necessitating an expensive rewrite. Conversely, if the modules are integrated top-down, then the lower-level modules usually will not receive as thorough a testing as would be the case if the product were integrated bottom-up. There could be an intentionally erroneous input of data and the product could crash or error-handling capabilities may not be adequate for dealing with bad data if the product is to be run together with the client's currently installed software. The new product may have an adverse effect on the client's existing computer operations. Finally, the source code and all other types of documentation may be incomplete and internally inconsistent. Again, Faults in shrink-wrapped software usually result in poor sales of the product and huge losses for the development company. There are also risks involved for a company participating in alpha or beta testing. In particular, alpha test versions can be fault-laden, resulting in frustration, time wasting, and possible damage to databases.

2.1.7 Maintenance Phase

Once the product has been accepted by the client, any changes constitute maintenance. Maintenance is not an activity that is grudgingly carried out after the product has gone into operations mode. On the contrary, it is an integral part of the software process that must be planned for from the beginning. Shaw (2010), maintained that a major limitation of this phase that made it the most important phase in the software development lifecycle is that more money is spent on maintenance than on all other software activities combined. However, Mathias (2009), maintained that a common problem with maintenance stage is documentation, or rather a lack of it. He further stated that in the course of developing software against a time deadline, the original specification and design documents are frequently not updated and are, consequently, almost useless to the maintenance team. Other documentation such as the database manual or the operating manual may never be written, because management decided that delivering the product to the client on time was more important than developing the documentation in parallel with the software. In many instances, the source code is the only documentation available to the maintainer. The high rate of personnel turnover in the software industry exacerbates the maintenance situation in that none of the original developers may be working for the organization at the time when maintenance is performed. By so doing the required changes may not have been correctly implemented.

2.1.8 Retirement Phase

The final phase in the software life cycle is retirement. After many years of service, a stage is reached when further maintenance is not cost-effective. Sometimes the proposed changes are so drastic that the design as a whole would have to be changed. In such a case it is less expensive to redesign and recode the entire product. Or perhaps so many changes may have been made to the original design that interdependencies have inadvertently been built into the product, and there is a real danger that even a small change to one minor module might have a drastic effect on the functionality of the product as a whole. Third, the documentation may not have been adequately maintained, thus increasing the risk of a regression fault to the extent that it would be safer to recode than to maintain. A fourth possibility is that the hardware (and operating system) on which the product runs is to be replaced; it may be more economical to rewrite than to modify. In each of these instances the current version is replaced by a new version, and the software process continues.

2.2 Number of Faults

Another major issue of plan-driven development is the significant number of faults found during the testing phase. Because testing is done only during the final stage of development, it is the first sacrificed when earlier phases of the process model take an unexpectedly long amount of time. In order to meet a fixed deadline, not enough testing can be done to find a sufficient amount of faults and thus the test coverage is low. Because it is usually the case that the project manager is overzealous in estimating task duration times and implementation does not elapse without a hitch, many faults persist when the product is released, which results in decreased customer satisfaction. Such Low customer satisfaction can reduce repeat or future sales, which decreases profits and thus the return on investment. Even when the planned amount of time for testing is available, it is difficult to find every fault since the entire code base needs to be examined at once. Also, since the quality of the implemented code is not known until just prior to the release date, when the functionality is tested, an unseen amount of faults may be found, which may be very expensive to fix. Thus this issue of plan-driven development also increases costs and drives down the profitability of a project. These now call for the need for the Agile method.

2.3 Limitations of Traditional Methods

According to Brooks (2010), the most common issues associated with a traditional method of software development include obsolete requirements, that is, if the requirements are unclear, incomplete, too general, or not testable. Brooks maintained that a lack of an opportunity to understand changing, current customer needs is a problem. In other words, if developers don't know what is needed or customers have erroneous expectations, problems are guaranteed. Again, inadequate testing and a high number of faults found during testing. No one will know whether or not the program is any good until the customer complains or systems crashes. If too much work is crammed in too little time, problems are inevitable. Harel, (2008), Stephenson, (2009), and Schach, (2015), itemized the various traditional Models of software development lifecycle and their various limitations as thus:

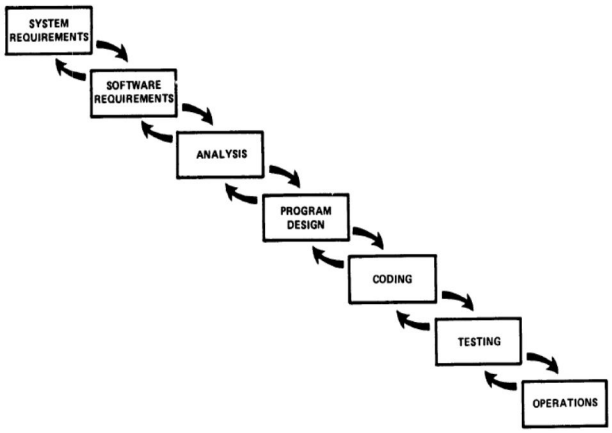
2.4 Traditional Models

There are numerous development models for developing the software in different paradigms. Some of these models are explained as below:

1) Waterfall Model:

It includes system, requirements, initial and detailed design, implementation, testing, operations, and maintenance. In the Boehm-Waterfall software engineering methodology the process interchanges from stage to stage. A corporation has involvement in building accounting systems, I/O controllers, and then building another such product based on the current enterprises is best managed with the waterfall model.

Fig 1: Waterfall SDLC Model

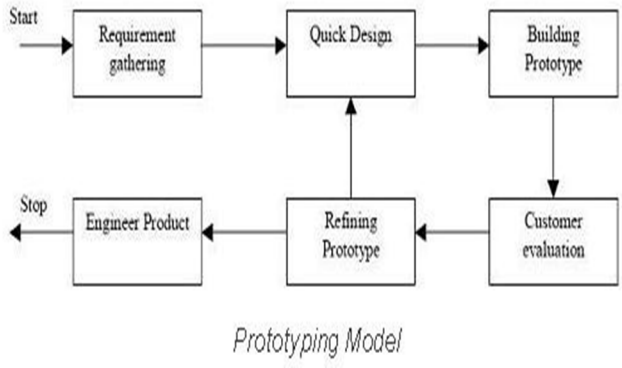


Waterfall model are lacked in: Risk requirement put on hold until late stage. Document confirmation is done late and also change in requirements was taken adversely in waterfall. Operational problems exposed too late in the process (acceptance testing), alteration was based on functionality, not on quality features.

2) Prototyping Model:

Prototyping is the process of building a working replica of a system. And it may be used with the waterfall, to create technical feasibility when technical risk is too high. It can also be used to better knowledge and extractment of user requirements. The basic goal is limitation of cost by understanding the problem before binding enough resources.

Fig 2: Rapid prototype SDLC Model



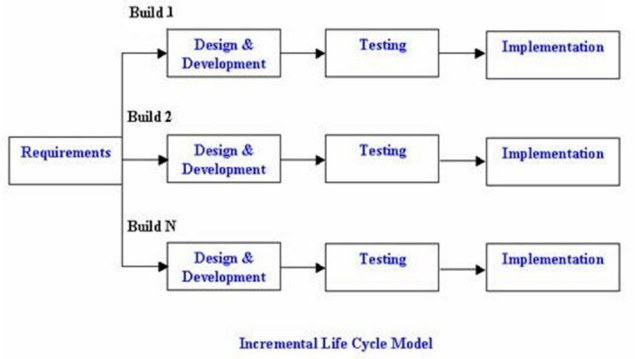
Prototyping Model

Prototyping model is lacked in: It is generally applicable to new and more original system and less to current one. Sometimes it can also produce a system with not a very good performance. It leads to implementing and then repairing way of building systems. Practically, this methodology may increase the complexity of the system as scope of the system may expand beyond original plans. Incomplete application may cause application not to be used as the full system was designed.

3) Incremental Model:

In incremental model the whole requirement is divided into various builds. Multiple development cycles take place here, making the life cycle a "multi-waterfall" cycle. Cycles are divided up into smaller, more easily managed modules. Each module passes through the requirements, design, implementation and testing phases. A working version of software is produced during the first module, so you have working software early on during the software life cycle. Each subsequent release of the module adds function to the previous release. The process continues till the complete system is achieved.

Fig 3: Incremental model of SDLC Model

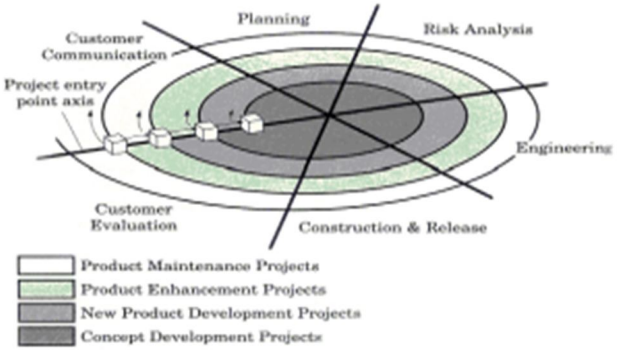


For example: In the diagram above when we work **incrementally** we are adding piece by piece but expect that each piece is fully finished. Thus keep on adding the pieces until it's complete. As in the image above a person has thought of the application. Then he started building it and in the first iteration the first module of the application or product is totally ready and can be demoed to the customers. Likewise in the second iteration the other module is ready and integrated with the first module. Similarly, in the third iteration the whole product is ready and integrated. Hence, the product got ready step by step.
Incremental model is lacked in: It needs good planning and design. It also needs a strong and complete description of the whole system before it can be broken down and built incrementally. The total cost is higher in comparison to waterfall.

4) Spiral Model:

Spiral model is an evolutionary software process model which is a combination of an iterative nature of prototyping and controlled and systematic aspects of traditional waterfall model. This model was originally proposed by Boehm. It provides the potential for rapid development of incremental versions of the software. In Spiral model, software development takes place in series of developed releases. In initial stage iterations, the release or model might be a paper model or a prototype. In the later stages a more complete version of software is actually produced. A spiral model is made up of set different framework activities made by the software engineering team. Take a look at the following spiral model diagram :

Fig 4: Spiral model of SDLC Model



The spiral size agrees to system size, while the detachment between the coils of the spiral indicates resources. It has basically four phases: Preparation, Risk Analysis, Engineering and Calculation. It is basically useful in ADE projects, as they are risky in nature. Business projects are more outmoded and generally use developed technology so this model is basically applicable to business applications, particularly in cases where there is no guaranteed success and also where applications require large calculations, example in decision support systems. During the risk examination phase, a process is undertaken to classify risk and substitute solutions, and at the end of which a prototype is produced. In the spiral model, progress is characterized by the pointy component, and the cost is signified by the radius of the spiral.

Spiral model is lacked in: It is a costly model to use. Risk analysis requires people with a particular expertise as the success of the project is dependent on the risk analysis phase. It does not work well for smaller projects.

2.5 Agile Methodology

Agile software development (also called "agile") isn't a set of tools or a single methodology, but a philosophy put to paper in 2001 with an initial 17 signatories, Igbokwe (2014). Agile was a significant departure from the heavyweight document-driven software development methodologies—such as waterfall—in general use at the time. While the publication of the "Manifesto for Agile Software Development" didn't start the move to agile methods, which had been going on for some time, it did signal industry acceptance of agile philosophy. A recent survey conducted by Dr. Dobb's Journal shows 41 percent of development projects have now adopted agile methodology, and agile techniques are being used on 65 percent of such projects.

As the development for the traditional software development methods started to fail, Agile software development methods (ASDMs) were developed. The determination towards agile started in the mid-1990s. This methodology can be described as iterative and incremental development including flexibility throughout the systems development life cycle. Minimal groundwork, light and fast development cycles, people-centric development, customer relationship, and regular delivery are the features of the Agile method. In 2001, seventeen experts met at Snowbird, Utah, to discuss if there was anything in common among the various agile methods (Cockburn, 2007), and they formed the Strategy for Agile Software Development (Becket al., 2001).

Highest priority is to accomplish the customer demands early and provide continuous delivery of esteemed software. It also appreciates the changing requirements, even if it affects the development process and the most proficient and working method of freeing information to and within a development team. Working software is the primary measure of any progress. Agile actions endorse maintainable progress. The best styles, requirements and designs appear from self-organizing teams. At continuous breaks, the team reflects on how to become more effective, then tunes and adjusts its performance accordingly.

There are twelve principles behind agile philosophy: they are

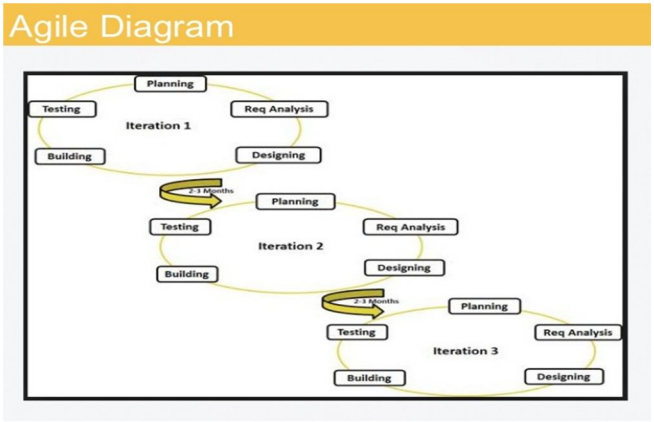
1. Our highest priority is to satisfy the customer through early and continuous delivery of products.
2. Appreciate changing necessities, for the user modest benefit.
3. Deliver working software frequently, within few days with a preference to the shorter timescale.
4. Business customers and developers must work together on daily basis throughout the project.
5. Projects should be developed around motivated people and they should be provided by the atmosphere and also by the support and trust to get the work completed.
6. The most 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 endorse supportable enlargement. The guarantors, designers, and employers should be able to maintain a continuous step open-endedly.
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 constructions, necessities, and strategies combined from self-organizing squads.
12. At regular intervals, the team reflects on how to become more efficient, then adjusts its behaviour accordingly.

2.5.1 Agile vs. Traditional Methods: practical differences in methodology

The traditional software development methodologies were hardly methodologies at all, but a free-for-all as organizations struggled to profit from new computer-related technologies. As the industry learned more about developing software, certain techniques for managing and predicting the cost of software development projects came into use. The methodology that has dominated software development projects for decades is called traditional methods. Winston Royce coined the term in 1970 to describe a serial method for managing software projects through the five stages- Requirement, Design, Implementation, Verification, Maintenance.

Adoption of the traditional methods has helped drive down the failure rate of software development projects, but even with rigorous project management and processes, a full 70 percent of software projects using this methodology fail to meet their objectives. To put this in perspective, traditional methods of software development projects have less than half the success rate (66 percent) of going over Niagara Falls in a barrel. Organizations tried to cut the failure rate by insisting on more detail in the requirements and design phases. This process of requiring extensive, even exhaustive, documentation culminated in 1988 with the publication of the Department of Defense Standard for software development, Jha (2011). One of the most important differences between the agile and Traditional approaches is that traditional features has distinct phases with checkpoints and deliverables at each phase, while agile methods have iterations rather than phases. The output of each iteration is working code that can be used to evaluate and respond to changing and evolving user requirements. Traditional methods assume that it is possible to have perfect understanding of the requirements from the start. But in software development, stakeholders often don't know what they want and can't articulate their requirements. With traditional method, developers rarely deliver what the customer wants even if it is what the customer asked for. Agile methodologies embrace iterations. Small teams work together with stakeholders to define quick prototypes, proof of concepts, or other visual means to describe the problem to be solved. The team defines the requirements for the iteration, develops the code, and defines and runs integrated test scripts, and the users verify the results. (See Figure 5.) Verification occurs much earlier in the development process than it would with waterfall, allowing stakeholders to fine-tune requirements while they're still relatively easy to change

Fig 5: Agile Methods of Software Development



2.5.2 Limitations of Agile Methods

Although agile development is not without its own issues, the number and severity of issues that impact costs and return on investment of a plan-driven methodology greatly outweigh those of an agile methodology, Robertson (2009). Common issues remaining for agile after the switch from traditional methods are related to high testing lead times, low test coverage, and many teams requiring high coordination and communication from project managers. An agile process model also does not scale well to large projects, as numerous iterations are needed to complete the desired functionality and too much time may be devoted to any single, small feature. Thus the opportunity cost of foregone production on more profitable and lean projects may be too high to employ agile methods on a large-scale project.

2.5.2 Testing Lead Times

High testing lead times can exist when a piece of functionality is only realized and implemented too late in an iteration to be tested and must wait for the next iteration to be tested. When switching to agile from a plan-driven method, such an issue is negligible compared to the excessive testing lead times of the latter, seeing as testing is saved for just prior to release (which may be several months after conception).

2.5.3 Test Coverage

Because continuous testing is needed in an agile methodology (testing at the end of every iteration), a substantive test environment containing test cases for every incremental piece of functionality is required. Agile development inherently requires that functionality be implemented in small steps with great attention to detail and current customer needs. This implies that testing must be done in small steps; it follows that such detailed implementation necessitates detailed testing. Thus more test cases are developed than in a plan driven approach as testing is done continuously and exhaustively, which suggests that testing is the bottleneck of an agile process model, Robertson (2009). Even though many test cases are developed, they tend to test on a unit level since small pieces of functionality are tested at a time. Because integration and system testing may be ignored more often than in a traditional methodology and there are simply many more test cases, testing in agile may result in low test coverage. As previously stated, low test coverage causes an increase in the number of faults found after the release of the product or more resources allocated to fixing such faults. Therefore, increased costs reduce profit and ROI.

2.5.4 Communication & Coordination

The last common issue with agile development involves management overhead. Because a successful application of an agile methodology relies heavily on strong teamwork, the project manager must remain involved in the dynamics of the team to foster a sense membership and attachment to the quality of the final product. This involves much communication and collaboration between team members and management to ensure all employees across process phases (business analysts, software developers, and testers) are working together to build a successful product. Also, the project manager must make certain that the incentives of all members are aligned with the mission of the project. In addition to such team-specific attention, agile development usually employs multiple teams working on different features of a product, especially for larger projects. It follows that management must provide such quality attention to multiple teams, each with different needs and behaviors. Although this issue does not directly affect costs, the time and effort needed to manage these teams may impact the resources available to work on other projects that do not employ an agile process model.

Although agile development does exhibit some issues, they pale in comparison to those of plan-driven development. Therefore, due to the most common issues of both methodologies, as presented by Kai Petersen and Claes Wohlin (2009) in their case study at Ericsson AB and based on my own analysis of their impact on several business metrics, agile development does the least harm to potential costs and return on investment.

2.6 Advantages of Agile Development methods

2.6.1 Stakeholder Engagement

Agile provides multiple opportunities for stakeholder and team engagement – before, during, and after each Sprint. By involving the client in every step of the project, there is a high degree of collaboration between the client and project team**,** providing more opportunities for the team to truly understand the client's vision. Delivering working software early and frequently increases stakeholders' trust in the team's ability to deliver high-quality working software and encourages them to be more deeply engaged in the project.

2.6.2 Transparency

An Agile approach provides a unique opportunity for clients to be involved throughout the project, from prioritizing features to iteration planning and review sessions to frequent software builds containing new features. However, this also requires clients to understand that they are seeing a work in progress in exchange for this added benefit of transparency.

2.6.3 Early and Predictable Delivery

By using time-boxed, fixed schedule Sprints of 1-4 weeks, new features are delivered quickly and frequently, with a high level of predictability. This also provides the opportunity to release or beta test the software earlier than planned if there is sufficient business value.

2.6.4 Predictable Costs and Schedule

Because each Sprint is a fixed duration, the cost is predictable and limited to the amount of work that can be performed by the team in the fixed-schedule time box. Combined with the estimates provided to the client prior to each Sprint, the client can more readily understand the approximate cost of each feature, which improves decision making about the priority of features and the need for additional iterations.

2.6.5 Allows for Change

While the team needs to stay focused on delivering an agreed-to subset of the product's features during each iteration, there is an opportunity to constantly refine and reprioritize the overall product backlog. New or changed backlog items can be planned for the next iteration, providing the opportunity to introduce changes within a few weeks.

2.6.6 Focuses on Business Value

By allowing the client to determine the priority of features, the team understands what's most important to the client's business, and can deliver the features that provide the most business value.

2.6.7 Focuses on Users

Agile commonly uses user stories with business-focused acceptance criteria to define product features. By focusing features on the needs of real users, each feature incrementally delivers value, not just an IT component. This also provides the opportunity to beta test software after each Sprint, gaining valuable feedback early in the project and providing the ability to make changes as needed.

2.6.8 Improves Quality

By breaking down the project into manageable units, the project team can focus on high-quality development, testing, and collaboration. Also, by producing frequent builds and conducting testing and reviews during each iteration, quality is improved by finding and fixing defects quickly and identifying expectation mismatches early.

During Segue's own experience of adopting Agile software development practices, we have seen solutions delivered on time and with a higher degree of client and customer satisfaction. By incorporating the ability to change, we have been able to better incorporate feedback from demos, usability testing, and client and customer feedback.

Agile is a powerful tool for software development, not only providing benefits to the development team, but also providing a number of important business benefits to the client. Agile helps project teams deal with many of the most common project pitfalls (such as cost, schedule predictability and scope creep) in a more controlled manner. By reorganizing and re-envisioning the activities involved in custom software development, Agile achieves those same objectives in a leaner and more business-focused way.

2.7 Degree of Agility

Thus far I have explored the benefits of switching to an agile methodology from a traditional process model under the assumption that the software development team employs all artifacts and values of an agile process. I will now question if the degree of agility applied by a firm affects the magnitude of such benefits.

Before I explore in detail the advantages (or lack thereof) of the various tools supplied by an agile methodology (for example Daily Scrum Meeting or Open Office Space), I would like to note that a team can only benefit from learning how to better understand and adopt agile methods in general. Maria Paasivaara (2011), and Casper Lassenius (2013) performed a case study, using an agile coaching team to instruct development teams in the use of an agile methodology, to illustrate the gains in productivity from being "more agile". They did not specify which tools are more beneficial than others, only that a more complete understanding of the agile philosophy can increase product quality and decrease overall project costs. The most important component of an "agile mindset" is teamwork. Improving the team dynamic generates many benefits: not only would members experience a sense of belonging at their place of work, but also feel free to discuss difficult aspects of the development process with others (which there are numerous). The free flow of information and discussion between team members facilitates both team and individual growth. As

individuals share experiences with others, both personal and work-related, a sense of trust is fostered throughout the team. The team then develops into a functional and productive one since all members are working together toward a common purpose. By learning from the experiences of others, individuals advance their personal growth as they cultivate such knowledge when away from the team. In order to gain these benefits in productivity (which increases ROI) from such team and individual growth, the team must embrace an agile mindset and understand that strong teamwork is very important in the success of the project, Jaana Nyfjord (2015). Mira Kajko- Mattsson (2012) have discovered through interviews with firms that employ agile practices that upfront, thorough planning allows for more agility in the implementation and testing phases. This suggests the use of a plan-driven approach during the pre-implementation phase of development. The degree of agility that was observed in this phase depended on many factors including project type and size, criticality, degree of uncertainty, budget, and the innovative character of the project. For the majority of fixed budget projects, a more traditional approach was employed before implementation because the development team could not afford to change the focus of the project based on current customer needs. At the same time, it is hard to conduct upfront, very thorough planning for small, innovative, and completely new projects. More agility in the pre-implementation phase is therefore required. The authors note that any degree of agility after performing some solid planning is acceptable as long as the project vision is not lost. Seeing as 60% of software companies do not subscribe to only one process model, few firms adopt an agile methodology in its entirety Obus (2007). Various fragments of agile development are utilized to help improve some failing aspect of the current process. For example, the Daily Scrum Meeting, a critical tool espoused by an agile methodology, can lead to increased collaboration, real-time information exchange, increased leadership and morale, and elevated communication. An open office space promotes faster problem solving and a reduced need for documentation but may decrease focus on work. Pair programming lends to improved product and design quality, reduced code defects, and increased creativity. Risk is reduced per iteration as a result of time-boxing, a planning technique that divides the schedule into several separate time periods, each containing its proper deliverables, deadline, and budget. Thus many agile techniques possess great advantages and help to reduce costs, but some, such as maintaining an open office space, may detract from the project goals. To conclude, a high degree of agility is beneficial only if the selected fragments do not cause more harm than good.

3. Methodology

I will now start to explain my analysis on data collected from various individuals with experience at firms that moved to an agile methodology. I hypothesized that the switch to agile indeed reduced costs, in line with my research.

3.1 Data Collection

I developed a questionnaire in other to gauge the effectiveness of an agile method versus a more traditional approach from experienced employees. My questionnaire sought to investigate the day-to-day experiences associated with the factors of agile development that reduce cost.

Not only did I desire to gain insight about the firm's agile practices, but also about its plan-driven process before the switch. Learning about the issues the firm experienced with a more traditional approach would help to understand the appeal of agile.

By collecting first-hand information about the work effects of a traditional versus an agile methodology, I can better understand from real world data the benefits of a switch to agile and make a conclusion regarding the success of the process model.

3.2 Data Analysis

Introductory emails were sent to University of Delhi India and Amity University India that recently moved to an agile process model. I received responses from a few alumni, after which I sent my questionnaire for completion. I believe this foot-in-door technique resulted in the highest probability of response. However, after sending out the questionnaire to those interested alumni, I received usable feedback from only one individual from Microsoft. My initial analysis was based on this piece of data.

4. Results

I wished to determine whether the observed experience with an agile methodology matched my research. Indeed, Microsoft does not use an agile process throughout its entire development cycle. Also, due to resource and time constraints, some teams must stop accepting faults and begin new iterations as necessary. The alumnus stated that team members feel more tied to projects under an agile process and desire to ensure that customer needs are met. Due to these experiences, it would seem that costs are reduced as a result of an enhanced sense of belonging and strong teamwork. At the same time, the alumnus experienced less time to react to faults later in the cycle under a traditional methodology, which also agrees with my research. Less available time to work on faults increases the number of defects found in the released product, which decreases ROI – (Return on Investment).

5. Conclusion and Future scope

A discussion on various enlargement models has been obtainable in this paper. Although many development models exist, this paper discusses different models out of them and the comparison including the advantages and disadvantages of different models which can help to select specific model at specific situations depending on customer request and business requirements. Agile method is taking the lead in software development as more and more software projects engage agile methods. There are also emerging patterns of success and failure. With growing adoption of agile methods, project managers increasingly need to understand the applicability to their projects and factors that drive key project performance characteristics. While some organizations affirm that agile methods solve all their problems, few have shown consistent success over a range of typical software projects. Agile methods have advantages, especially in accommodating change due to volatile requirements. However, they also present concomitant risks with managing the many dependent pieces of work distributed across a large project. Use of agile methods therefore presents a set of tradeoffs. This paper examines the impact of agile methods on the people involved in a project, the process under which a project is developed, and on the project itself in an attempt to allow project managers to evaluate the applicability of using an agile method. My results will not produce any surprising points, but with time and a greater response rate from employees that are exposed to agile methods, more substantial results may be found. In the future, much more data may be collected to generate more extensive analyses. The increases or decreases in costs and return on investment may also be quantized to provide more quantitative evidence of the benefits of agile development.

5.1 Recommendations for further studies

In order to achieve comparable future breakthroughs, brooks suggest that we change the way that software is produced. Brooks, 2006. For example, whenever possible, software products should be bought off the shelf (that is, shrink-wrapped software) rather than custom-built. For brooks, the hard part of building software lies in the requirements, specification, and design phases, not in the implementation phase. The use of agile method is for him a major potential source of an order-of- magnitude improvement.

Another suggestion that, in Boehm and Papaccio's (2010) opinion, may lead to a major improvement in productivity is greater use of the principle of incremental development, where instead of trying to build the product as a whole, it is constructed stepwise.

For Card, Mcgarry, and Page (2009), the greatest hope of a major breakthrough in improving software production lies in training and encouraging great designers. As stated previously, in Beizer's (2014) opinion, one of the hardest aspects of software production is the design phase, and to get great designs, we need great designers. Goldberg, (2009) cites unix, apl, pascal, modula-2, smalltalk, and fortran as exciting products of the past. He points out that they have all been the products of one, or a very few, great minds.

On the other hand, more prosaic but useful products like cobol, plii, algol, mvs/360, and ms-dos have all been products of committees. Nurturing great designers is for brooks the most important objective if we wish to improve software production.

REFERENCES

Beizer and Heizer. 2014, Software Testing Techniques, Second Edition, Van Nostrand Reinhold, New York, NY, 2014.

Boehm, 2010., 'Software Engineering," IEEE transactions on computers c-25 (December 2010), pp. 1226—1241.

Boehm and Papaccio, 2010. "Understanding and Controlling Software Costs," IEEE transactions on software engineering 14 (October 2010), pp. 1462—1477.

Brooks, 2006 f. P. Brooks, Jr.. The Mythical Man-Month: Essays on Software Engineering, Addison-Wesley, Reading, MA, 2006.

Brooks, 2006 f p. Brooks, Jr., "no silver bullet," in: information processing. H.-j. Kugler (Editor), Elsevier North-Holland, New York, NY, 2006. Reprinted in: IEEE computer (April 2006), pp. 10—19.

Card, Mcgarry, and Page, 2009. "Evaluating Software Engineering Technologies," IEEE transactions on software engineering SE-13 (July 2007), pp. 845—85 1.

Goldberg, 2009. "Software Engineering: an emerging discipline," IBM Systems Journal 25 (no. 3/4, 2009), pp. 334—353.

Kruchten, Philippe. *A Plea for Lean Software Process Models*. Rep. 2011. Print.

Mathis, 2010 "the last 10 percent" IEEE transactions on Software Engineering SE-12 (June 2010), pp. 705—712.

Nyfjord, Jaana, and Mira Kajko-Mattsson. *Degree of Agility in Pre-implementation Process Phases*. Rep.Berlin: Springer-Verlag, 2008. Print.

Paasivaara, Maria. *How Does an Agile Coaching Team Work?: A Case Study*. Rep. New York: ACM, 2011.Print.

Petersen, Kai. *The Effect of Moving from a Plan-Driven to an Incremental Software Development Approach with Agile Practices*. Rep. Print.

Petersen, Kai. *An Empirical Study of Lead-Times in Incremental and Agile Software Development*. Rep. Springer, 2010. Print.

Ronkko, Mikko, Antero Jarvi, and Markus M. Makela. *Measuring and Comparing the Adoption of Software Process Practices in the Software Product Industry*. Rep. Berlin: Springer, 2008. Print.