2012 International Conference on Information and Network Technology (ICINT 2012)
IPCSIT vol. 37 (2012) © (2012) IACSIT Press, Singapore

Software Development Life Cycle AGILE vs Traditional Approaches

Yu Beng Leau [+], Wooi Khong Loo, Wai Yip Tham and Soo Fun Tan

School of Engineering and Information Technology Universiti Malaysia Sabah, Malaysia

Abstract. Software development life cycle is the most important element in software development. It depicts the necessary phases in software development. This paper reviews the modern SDLC which are traditional methods and agile methods and explains the advantages and disadvantages of both methodologies. It also suggests improvements for current agile development so that this lightweight SDLC could be adopted more in practice for organizational project management.

Keywords: SDLC, agile, traditional methods, ROI.

1. Introduction

Software Development Life Cycle (SDLC) is a process of building or maintaining software systems[7]. Typically, it includes various phases from preliminary development analysis to post-development software testing and evaluation. It also consists of the models and methodologies that development teams use to develop the software systems, which the methodologies form the framework for planning and controlling the entire development process.

A software application or an information system is designed to perform a particular set of tasks. Often, this set of tasks that the system will perform provides well-defined results, which involve complex computation and processing. It is therefore a harsh and tedious job to govern the entire development process to ensure that the end-product comprises of high degree of integrity and robustness, as well as user acceptance. Thus, a systematic development process which is able to emphasize on the understanding of the scope and complexity of the total development process is essential to achieve the said ¬characteristics of a successful system.

Currently, there are two SDLC methodologies which are utilized by most system developers, namely the traditional development and agile development which explained in next session. In section 4 we compare and contrast these two methodologies in detail and suggested some improvements in following section. Finally, the conclusion is presented.

2. Traditional Software Development

Software methodologies like Waterfall method, V-Model and RUP are called traditional software development methodologies and these are classified into the heavyweight methodologies [1]. These methodologies are based on a sequential series of steps like requirements definition, solution building, testing and deployment. Traditional software development methodologies require defining and documenting a stable set of requirements at the beginning of a project.

There are four phases which are characteristic of traditional software development method. The first step is to set up the requirements for the project and determine the length of time it will take to implement the various phases of development while trying to predict any problems that may arise in the project. Once the requirements are laid out, the next step moves into the design and architectural planning phase where a

[+] Corresponding author. Tel.: + 6088320000 ext 3220; fax: +6088320348.
E-mail address: lybeng@ums.edu.my.

technical infrastructure is produced in the form of diagrams or models. These bring to the surface potential issues that the project may face as it progresses and provide a workable road map for the developers to implement.

Once the team is satisfied with the architectural and design plan, the project moves into the development phase where code is produced until the specific goals are reached. Development is often broken down into smaller tasks that are distributed among various teams based on skill. The testing phase often overlaps with the development phase to ensure issues are addressed early on. Once the project nears completion and the developers are close to meeting the project requirements, the customer will become part of the testing and feedback cycle and the project was delivered after the customer satisfy with it.

The traditional software development methods are dependent on a set of predetermined processes and on-going documentation which is written as the work progresses and guides further development [1]. The success of a project which is approached in this way relies on knowing all of the requirements before development begin and means that implementing change during the development lifecycle can be somewhat problematic. However, it also makes it easier to determine the costs of the project, set a schedule and allocate resources accordingly [2].

3. AGILE Software Development

Agile development is based on the idea of incremental and iterative development, in which the phases within a development life cycle are revisited over and over again. It iteratively improves software by using customer feedback to converge on solutions[5].

In agile development, rather than a single large process model that implemented in conventional SDLC, the development life cycle is divided into smaller parts, called “increments” or “iterations”, in which each of these increments touches on each of the conventional phases of development. According to Agile Manifesto, the major factors of agile factors include the following four:

1. Early customer involvement
2. Iterative development
3. Self-organizing teams
4. Adaptation to change

There are currently six methods that are identified as agile development methods, which are Crystal methodologies, dynamic software development method, feature-driven development, lean software development, scrum, and extreme programming[8].

4. Discussion

One major difference between agile development and conventional development methods is that the former methodology possesses the ability to successfully deliver result quickly and inexpensively on complex projects with ill-defined requirements. Agile methods emphasize on teams, working software, customer collaboration, and responding to change; while the conventional methods stress on contracts, plans, processes, documents, and tools.

Agile development methods take business Return of Investment (ROI) as its utmost priority[11]. In traditional development life cycle, the development teams usually hold a meeting with the stakeholders and obtain every detailed requirement during early phases of development process. Then the development teams would start the design phase, followed by the actual coding phase. The testing phase will only start when the entire coding process is completed. Then only will the end-product be presented to stakeholders after there is no issue arises in the testing phase. The shortcoming of this traditional methodology is that the development teams build the system in “one-shot” fashion. Assume that an issue arises during the testing phase, the worst case scenario would be the entire module would have to be reverted to rectify the issue. Another problem with the traditional SDLC is that in most cases, stakeholders will not know what they really want to implement in the system, therefore the requirement model engineered at the earlier phases might not necessarily be the actual features that need to be implemented. Users’ or stakeholders’ change requests might flow in after the end-product is presented and released on the market. This would further speed up the deterioration of the software as multiple change requests from different parties that implemented into the system would cause various compatibility and software integrity issues. These problems are even more apparent in larger system. Hence, from a business perspective, the traditional SDLC is not an adequately efficient methodology.

In contrast to traditional SDLC, the Agile SDLC avoids ‘up-front’ requirement gathering as stakeholders often could not provide all requirements in sufficient details for implementation to occur at the beginning of a project[13]. It is a common phenomenon that customers could not decide the features to be included in the system. Therefore, the frequent demonstration and release of software in common agile practices allow customers to acquire sufficient details on the current release of the system upon actual interaction with system and thus providing feedback to refine the requirements provided earlier before the current release. The iterative approach in agile practices also allows customers to delay decisions, where decisions may be delayed to some future iteration when better information or technology is available to optimize the choice. It is also the one of the advantages that agile SDLC triumphs traditional SDLC by the fact that in agile SDLC, development can begin even before all the requirements are known.

Taking the fact that customers’ requirements are acquired iteratively as context, agile development is able to deliver an end-product that better meets customer needs. For every short run of iteration, completed modules are presented to customers for review. These modules are by no means integrated as a full system and thus any rework or additional features would not marginally increase the development cost. Taking this advantage developers are always ready to include any features that customer desire, and the system integration will only occur when customers have no further additional requirements. Apparently, this approach could greatly satisfy customers with a complete system containing all desired functions.

It is also highly possible for stakeholders to maximize their business return on investment by practicing agile methods in system development. The direction of the development is always readily changeable and the cost of change is low, as the stakeholders are given the opportunity to revise the business factors at the beginning of each iteration to include additional features into the system according to business ROI[11]. However, it is also the responsibilities of the development team to inform the stakeholders of the technical risk of the change. This attribute of agile methodology is known as modular and lean which allows mobility of particular features or components in the system or process depending on specific needs of stakeholders.

Although Agile methodologies triumph traditional methodologies in many aspects, there exist several difficulties in putting it into practice. One among these is that agile methods significantly reduce the amount of documentation, and even claim that the code itself should act as a document[14]. This causes developers who are accustomed to agile methods have a tendency to place more comments in the code as explanation and clarification. However, it is difficult for novice developers or new team members to complete tasks when they could not adequately comprehend the project. They thus pose numerous questions for the experienced developers and this could cause the delivery of iteration to be delayed, which in turn may cause an increase in development cost. Traditional methods on the other hand, stress on the importance of documentation in providing guidelines and clarification on the project for development team, thus has no relevant concern of developers not being knowledgeable of the project detail or the availability of a knowledgeable developer.

Agile methodologies are well-known for emphasizing in communication and customer involvement[9]. For every deliverable iteration, the development team and customers will hold a meeting, where the team members will communicate and summarize their work done in this iteration; whereas customers will provide feedback on the delivered software to refine current features or include additional features in the system. Most of the time, developers will find the regular meetings, mostly on weekly basis, are tedious and tiring as they would have to present to other members and customers of their responsible modules repeatedly, and upon each iteration, various changes to the modules will most likely to happen due to change in requirements. Furthermore, the time frame allocated for each iteration is typically short, which usually in the range of weeks. Developers would often find that the schedule is tight for them to develop each of the modules, this is even more so if the particular module involves complicated processing algorithms. This draws the delivery of each iteration behind the schedule and thus an efficient communication between the team members and with the customers could not be established. On the other hand, traditional methodologies have a well-defined requirement model before the implementation and coding process starts, in which this model would act as a reference for development team during the coding process. Customers are not likely to participate in this phase of development life cycle, while development team will do the coding according to the documentation provided by business analysts until the entire system is completed and integrated, then only will the integrated system be presented to customers as end-product. In this case, developers will not have to concern about the frequent iteration meetings and could be allowed a wider time frame to complete the system, thus allowing them to provide a better result.

As mentioned previously, agile development focuses on communication and customer involvement, in which by this premise it implies that interpersonal and social skills are crucial for the entire development team so that during each iteration, the completed modules can be efficiently delivered to customers and enlighten them of the current progress of the development and, if there is any, issues that developers encountered during implementation and coding phase. It is also important for the development team to be fully understand about the requirements and changes proposed by customers, where this ultimately require effective communication skills. It is however, not every developer would possess good social skills. Whenever a developer within the team has poor social skill, relevant parties will have difficulties obtaining information on the particular module progress, where this in turn causes customers to be unable to provide accurate requirement for subsequent iteration. While developers could not understand what exactly is required by customers, it is very likely that the completed module contains unwanted features. Thus this further increases the cost of development by reworking the module, and the increased reliance on social skills of developers would increases the instability of the development process.

The fact that agile development open to incremental requirement changes has gave rise to two dependency issues in design, which are namely rigidity and mobility. Rigidity refers to a change in the system implies a cascade of changes in other modules; while mobility refers to inability of the system to encapsulate components that can be reused, because it implies too much effort or risk. If these issues are all over the system, high-level restructuring is required to remove unwanted dependencies[15]. One immediate consequence of these dependency issues is the violation of the Interface Segregation Principle [16], explaining most of the difficulties in the deployment stage. Table 1 depicts the differences of AGILE and traditional approaches in several aspects.

Table. 1: Comparison of Agile and Traditional Approaches

	AGILE	**TRADITIONAL**
User requirement	Iterative acquisition	Detailed user requirements are well-defined before coding/implementation
Rework cost	low	high
Development direction	Readily changeable	Fixed
Testing	On every iteration	After coding phase completed
Customer involvement	high	low
Extra quality required for developers	Interpersonal skills & basic business knowledge	Nothing in particular
Suitable Project scale	low to medium-scaled	Large-scaled

5. Improvement

Agile software development methods were developed to provide more customer satisfaction, to shorten the development life cycle, to reduce bug rates, and to accommodate changing business requirement during the development process[13]. It is a very useful methodology to be adopted in the modern software development process to replace the traditional heavyweight development life cycle. However, it is still not perfected yet and faces several barriers in putting it into practices.

From previous discussion it is revealed that agile methods promote communication over documentation, and this attribute of agile development reduces comprehension of the system. There is no guarantee that the code can serve as documentation if the system was originally developed using different methods. Therefore it is crucial to make the system comprehensive in order to improve its feasibility. This could be done by introducing automated code inspection, where source codes are checked for compliance with a predefined set of rules or best practices, by means of software tools. With this, code defects could be detected before the testing process begins, thus saving time and cost. It also provides complete code coverage and is able to identify defects that testing process might have missed, as well as the root cause of the defect. In addition, refactoring to untangle crosscutting concerns will also improve the comprehensibility as they help to distinguish code for various business segments and separate business and platform code. In order to improve the modifiability and deployability of the system, developers should always do an analysis of the module dependencies and specific refactoring to further improve these two interfaces.

Another important parameter in promoting the agile development in practice is the social skills of developers. Developers who engaged in an agile project development should have good social and interpersonal skills instill in them. This could be achieved by either training provided by company, or developers themselves should be well-aware of the importance of this quality and always be motivated to polish their interpersonal skills, by means of frequent communication and socialisation with people around. By communicate more with different people, one can grasp the technique to express themselves more effectively and also to understand what are other people trying to express.

Developers should as well equipped with basic business knowledge in order to speak with customers on equal basis. A team without business knowledge most likely will not able to deliver the product which is valuable to customers, thus causing customers to lose faith in the development team. Hence, it is important for developers not only excel in computer science knowledge, but also have basic understanding in business rules. This can be done by short training provided by company or by inviting domain expert to lift team members' knowledge to an upper level of understanding.

A few points should also be noted for developers to practice agile development. First of all, they should make incremental change to the requirement, project plan system, and the resulting artifacts to enable agility. They should also strive for feedback to ensure the project meets the needs of all participants and stakeholders. In addition, only those tasks that add value to business processes supported by the system should be performed and lastly, processes and artifacts that do not add enduring value to the working software system should be discarded.[12]

6. Conclusion

Software Development Life Cycle is a methodology that depicts the entire development process, in which a software development organization ought to utilize to ensure a successful software development. While modern SDLC are divided into two main categories, which are traditional SDLC and agile SDLC.

As discussed earlier, agile SDLC excels traditional SDLC. However, agile SDLC also has its disadvantages. While agile SDLC is more suitable for small-medium project development, it is still better to adopt traditional SDLC for large-scale project. Therefore, it is important that development team select a SDLC that best suits the project.

There some criteria that development team could use to identify the desired SDLC, these include size of team, geographical situation, size and complexity of software, type of project, business strategy, engineering capability, and others where it may be found appropriate. It is also crucial for the team to study the differences, advantages, and disadvantages of each SDLC before hammer down the decision. In addition, the team must study the business context, industry requirements, and business strategy to be able to assess the candidate SDLC against the selection criteria.

A SDLC selection and adoption process is crucial that it ensures the organization to maximize their chance to deliver their software successfully, therefore selecting and adopting the right SDLC is a management decision with long term implications.

7. References

[1] Nikiforova, O., Nikulsins, V., Sukovskis, U.: *Integration of MDA Framework into the Model of Traditional Software Development.* In: Frontiers in Artificial Intelligence and Applications, Databases and Information Systems V, vol. 187, pp. 229–239. IOS Press, Amsterdam (2009)

[2] IBM*: Rational Unified Process: Best practices for software development teams* (2003), http://www.ibm.com/developerworks/rational/library/253.html

[3] Fowler M. *The New Methodology*, 2005

[4] Wysocki R. K., McGary R., *Effective Project Management*, Third Edition, John Wiley & Sons © 2003

[5] Szalvay, Victor. *An Introduction to Agile Software Development*. Danube Technologies Inc. 2004.

[6] Cohen, S. *A Software System Development Life Cycle Model for Improved Stakeholders' Communication and Collaboration.* Int. J. of Computers, Communications & Control. Vol. V, No. 1, pp. 20-4. 2010.

[7] Systems *Development Lifecycle: Objectives and Requirements*. Bender RPT Inc. 2003.

[8] Dyba, Tore. *Empirical studies of agile software development: A systematic review*. 24 January 2008.

[9] Peterson, Kai. *A Comparison of Issues and Advantages in Agile and Incremental Development between State of the* Art *and an Industrial Case*. Journal of System and Software. 2009.

[10] Abrahamsson, Pekka. *Agile Software Development Methods: Review and Analysis*. Julkaisua-Utgivare-Publisher. 2002.

[11] Rico, David F. *What is the ROI of agile vs. traditional Methods*. 2008.

[12] Carayannis, E.G. *Agile Project Management for IT Project*. Greenwood Press / Quorum Books. 2002

[13] Cho, Juyun. *Issues and Challenges of agile software development with SCRUM*. Issues in Information System. VOL IX, No. 2. 2008.

[14] Vijayasarathy, Leo R. *Agile Software Development: A survey of early adopters*. Journal of Information Technology Management Volume XIX, Number 2. 2008.

[15] Henssen, Geir K. *Maintenance and Agile Development: Challenges, Opportunities and Future Directions*. 2009.

[16] Martin, R.C., *Agile Software Development, Principles, Patterns and Practice*. Prentice Hall. 2002.