RESEARCH ARTICLE OPEN ACCESS

Efficiency Analysis of Software Development Life Cycle Models

Pallab Banerjee [1], Biresh kumar [2], Amarnath singh [3]
Arundhati Singh [4], Rupsi Kumari [5]
[1]Assistant Professor [1], [2], [3], B.Tech Scholar [4], [5]
Department of Computer Science and Engineering
Amity University, Ranchi Jharkhand
India

ABSTRACT

The development lifecycle of software comprises of five major stages namely Feasibility study, Requirement Elicitation, Designing, Coding and Testing. A software process model is the basic framework which gives a workflow from one stage to the next. This workflow is a guideline for successful planning, organization and final execution of the software project. Generally we have many different techniques and methods used to software development life cycle. Project and most real word models are customized adaptations of the generic models while each is designed for a specific purpose or reason, most have similar goals and share many common tasks. This paper will explore the similarities and difference among these various software development life cycle models.

Keywords: Software Management Processes, Software Development Process, Software Development Life Cycle, Comparative analysis of Software development life cycle models.

I. INTRODUCTION

No one can deny the importance of computer in our life, especially during the present time. In fact, computer has become indispensable in today's life as it is used in many fields of life such as industry, medicine, commerce, education and even agriculture. The purpose of this paper is to provide an understanding of the Software Development Lifecycle (SDLC) models available to software developers. A Software Project is instructions (computer programs) that, when executed, provide desired features, function and performance; data structures that enable the program to adequately manipulate information and documents that describe the operation and use of the program. Software engineers have to face many challenges when they start developing a new software project like developing techniques to build software project that can easily cope with heterogeneous platforms and execution environment. There are various software development approaches defined and designed, which are used/employed during development process of a software, these approaches are also referred to as "Software Development Process Models". Each process model follows a particular life cycle in order to ensure success in the process of software development. Note that the SDLC acronym is also used to represent System Development Life Cycle. New SDLC models are introduced on a regular basis as new technology and new research requires new SDLC techniques. Recent new SDLC models include Extreme Programming and Agile Development. A Software Development Life Cycle Model is a set of activities together with an ordering relationship between activities performed in a manner that satisfies the ordering relationship that will produce desired product. A software development life cycle model is broken down into distinct activities and specifies how these activities are organized in the entire software development effort. This paper looks at the most commonly known and used models and compares their efficiency analysis.

II. PHASES INVOLVED IN SDLC MODELS

The phases that are generally present in each and every software development life cycle model are:

- Requirement Analysis: Requirement analysis is the most important and fundamental stage in SDLC. This information is then used to plan the basic project approach and to conduct product feasibility study in the economical, operational and technical areas.
- Design: A design approach clearly defines all the architectural modules of the product along with its communication and data flow representation with the external and third party modules (if any). The internal design of all the modules of the proposed architecture should be clearly defined with the minutest of the details in DDS (Design Document Specification).
- Implementation: In this stage of SDLC the actual development starts and the product is built. The programming code is generated as per DDS during this stage. If the design is performed in a detailed and organized manner, code generation can be accomplished without much hassle.
- Testing: This stage is usually a subset of all the stages as in the modern SDLC models, the testing activities

are mostly involved in all the stages of SDLC. However, this stage refers to the testing only stage of the product where product defects are reported, tracked, fixed and retested, until the product reaches the quality standards defined in the SRS (Software Requirement Specification).

- Evolution: Once the product is tested and ready to be deployed it is released formally in the appropriate market. Sometimes product deployment happens in stages as per the business strategy of that organization. The product may first be released in a limited segment and tested in the real business environment.

The common phases of an SDLC can be represented by the following diagram:

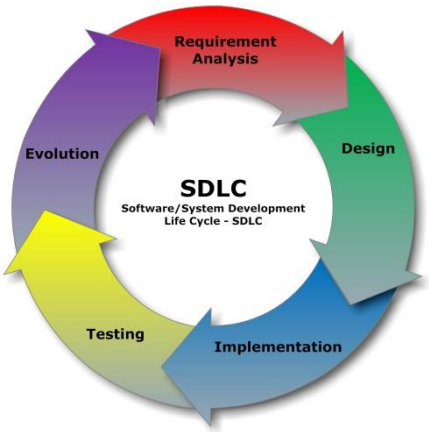

Fig. 1 Phases involved in SDLC Models

III. SDLC MODLELS

There are various software development life cycle models defined and designed which are followed during the software development process. Each process model follows a series of steps unique to its type to ensure success in the process of software development.

Following are the most important and popular SDLC models followed in the industry:

- Waterfall Model
- Iterative Model
- Spiral Model
- V-Model
- Big Bang Model
- Rapid Application Development Model
- Agile Model

A. Waterfall Model

Waterfall approach was first SDLC Model to be used widely in Software Engineering to ensure success of the project. In "The Waterfall" approach, the whole process of software development is divided into separate phases. In this Waterfall model, typically, the outcome of one phase acts as the input for the next phase sequentially.

The following illustration is a representation of the different phases of the Waterfall Model:

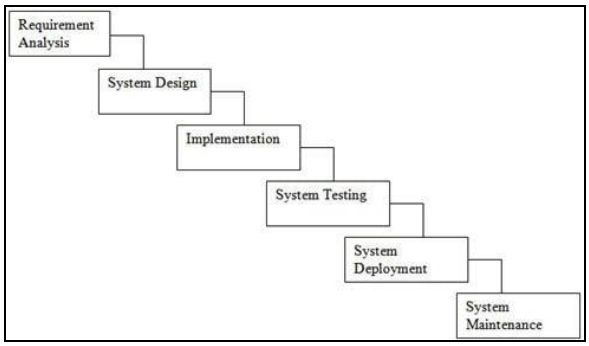

Fig 2. Waterfall Model

The sequential phases in Waterfall Model are:

- Requirement Gathering and Analysis - All possible requirements of the system to be developed are captured in this phase and documented in a requirement specification document.
- System Design - The requirement specifications from first phase are studied in this phase and the system design is prepared. This system design helps in specifying hardware and system requirements and helps in defining the overall system architecture.
- Implementation - With inputs from the system design, the system is first developed in small programs called units, which are integrated in the next phase. Each unit is developed and tested for its functionality, which is referred to as Unit Testing.
- Integration and Testing - All the units developed in the implementation phase are integrated into a system after testing of each unit. Post integration the entire system is tested for any faults and failures.
- Deployment of System - Once the functional and non-functional testing is done; the product is deployed in the customer environment or released into the market.
- Maintenance - There are some issues which come up in the client environment. To fix those issues, patches are released. Also to enhance the product some better versions are released. Maintenance is done to deliver these changes in the customer environment.

Waterfall Model – Application:

Every software developed is different and requires a suitable SDLC approach to be followed based on the internal and external factors. Some situations where the use of Waterfall model is most appropriate are:

- Requirements are very well documented, clear and fixed.
- Product definition is stable and short.
- Technology is understood and is not dynamic.
- There are no ambiguous requirements.
- Ample resources with required expertise are available to support the product.

Waterfall Model – Analysis:

Some of the major advantages of the Waterfall Model are as follows

- Simple and easy to understand and use
- Easy to manage due to the rigidity of the model. Each phase has specific deliverables and a review process.
- Phases are processed and completed one at a time.
- Works well for smaller projects where requirements are very well understood.
- Clearly defined stages.
- Well understood milestones.
- Easy to arrange tasks.
- Process and results are well documented.

The major disadvantages of the Waterfall Model are as follows

- No working software is produced until late during the life cycle.
- High amounts of risk and uncertainty.
- Not a good model for complex and object-oriented projects.
- Poor model for long and ongoing projects.
- Not suitable for the projects where requirements are at a moderate to high risk of changing. So, risk and uncertainty is high with this process model.
- It is difficult to measure progress within stages.
- Cannot accommodate changing requirements.
- Adjusting scope during the life cycle can end a project.
- Integration is done as a "big-bang. at the very end, which doesn't allow identifying any technological or business bottleneck or challenges early.

B. Iterative Model

Iterative process starts with a simple implementation of a subset of the software requirements and iteratively enhances the evolving versions until the full system is implemented.

At each iteration, design modifications are made and new functional capabilities are added. The basic idea behind this method is to develop a system through repeated cycles (iterative) and in smaller portions at a time (incremental).

The following illustration is a representation of the Iterative and Incremental model −

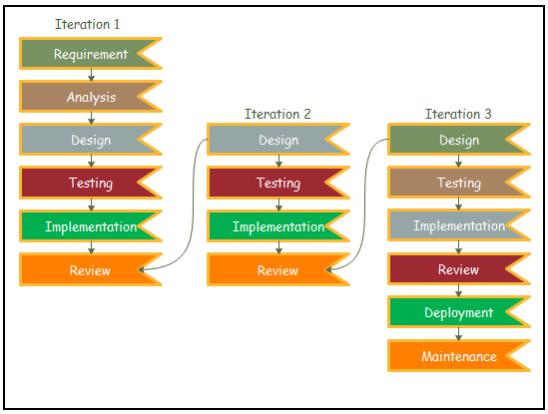

Fig 3. Iterative Model

In this incremental model, the whole requirement is divided into various builds. During each iteration, the development module goes through the requirements, design, implementation and testing phases. Each subsequent release of the module adds function to the previous release. The process continues till the complete system is ready as per the requirement. The key to a successful use of an iterative software development lifecycle is rigorous validation of requirements, and verification & testing of each version of the software against those requirements within each cycle of the model. As the software evolves through successive cycles, tests must be repeated and extended to verify each version of the software.

Iterative Model – Application:

Like other SDLC models, Iterative and incremental development has some specific applications in the software industry. This model is most often used in the following scenarios

- Requirements of the complete system are clearly defined and understood.
- Major requirements must be defined; however, some functionalities or requested enhancements may evolve with time.
- There is a time to the market constraint.
- A new technology is being used and is being learnt by the development team while working on the project.
- Resources with needed skill sets are not available and are planned to be used on contract basis for specific iterations.

- There are some high-risk features and goals which may change in the future.

Iterative Model – Analysis:

The advantages of the Iterative and Incremental SDLC Model are as follows

- Some working functionality can be developed quickly and early in the life cycle.
- Results are obtained early and periodically.
- Parallel development can be planned.
- Progress can be measured.
- Less costly to change the scope/requirements.
- Testing and debugging during smaller iteration is easy.
- Risks are identified and resolved during iteration; and each iteration is an easily managed milestone.
- Easier to manage risk - High risk part is done first.
- With every increment, operational product is delivered.
- Issues, challenges and risks identified from each increment can be utilized to the next increment.
- Risk analysis is better.
- It supports changing requirements.
- Initial Operating time is less.
- Better suited for large and mission-critical projects.

The disadvantages of the Iterative and Incremental SDLC Model are as follows

- More resources may be required.
- Although cost of change is lesser, but it is not very suitable for changing requirements.
- More management attention is required.
- System architecture or design issues may arise because not all requirements are gathered in the beginning of the entire life cycle.
- Defining increments may require definition of the complete system.
- Not suitable for smaller projects.
- Management complexity is more.
- End of project may not be known which is a risk.
- Highly skilled resources are required for risk analysis.
- Projects progress is highly dependent upon the risk analysis phase.

C. Spiral Model

The spiral model, initially proposed by Boehm, is an evolutionary software process model that couples the iterative feature of prototyping with the controlled and systematic aspects of the linear sequential model. It implements the potential for rapid development of new versions of the software. Using the spiral model, the software is developed in a series of incremental releases. During the early iterations, the additional release may be a paper model or prototype. During later iterations, more and more complete versions of the engineered system are produced.

The Spiral Model is shown in fig:

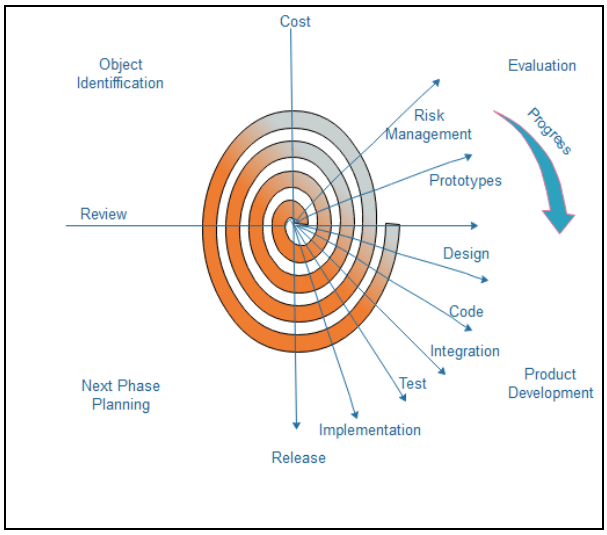

Fig 4. Spiral Model

The spiral model has four phases:

- Objective setting: Each cycle in the spiral starts with the identification of purpose for that cycle, the various alternatives that are possible for achieving the targets, and the constraints that exists.
- Risk Assessment and reduction: The next phase in the cycle is to calculate these various alternatives based on the goals and constraints. The focus of evaluation in this stage is located on the risk perception for the project.
- Development and validation: The next phase is to develop strategies that resolve uncertainties and risks. This process may include activities such as benchmarking, simulation, and prototyping.
- Planning: Finally, the next step is planned. The project is reviewed, and a choice made whether to continue with a further period of the spiral. If it is determined to keep, plans are drawn up for the next step of the project

Spiral Model – Application:

The following pointers explain the typical uses of a Spiral Model

- When there is a budget constraint and risk evaluation is important.
- For medium to high-risk projects.
- Long-term project commitment because of potential changes to economic priorities as the requirements change with time.
- Customer is not sure of their requirements which are usually the case.
- Requirements are complex and need evaluation to get clarity.
- New product line which should be released in phases to get enough customer feedback.
- Significant changes are expected in the product during the development cycle.

Spiral Model – Analysis:

The advantage of spiral lifecycle model is that it allows elements of the product to be added in, when they become available or known. This assures that there is no conflict with previous requirements and design.

This method is consistent with approaches that have multiple software builds and releases which allows making an orderly transition to a maintenance activity. Another positive aspect of this method is that the spiral model forces an early user involvement in the system development effort.

On the other side, it takes a very strict management to complete such products and there is a risk of running the spiral in an indefinite loop. So, the discipline of change and the extent of taking change requests is very important to develop and deploy the product successfully.

The advantages of the Spiral SDLC Model are as follows

- Changing requirements can be accommodated.
- Allows extensive use of prototypes.
- Requirements can be captured more accurately.
- Users see the system early.
- Development can be divided into smaller parts and the risky parts can be developed earlier which helps in better risk management.

The disadvantages of the Spiral SDLC Model are as follows

- Management is more complex.
- End of the project may not be known early.
- Not suitable for small or low risk projects and could be expensive for small projects.
- Process is complex
- Spiral may go on indefinitely.
- Large number of intermediate stages requires excessive documentation.

D. V-Model

The V-model is an SDLC model where execution of processes happens in a sequential manner in a V-shape. It is also known as Verification and Validation model. Under the V-Model, the corresponding testing phase of the development phase is planned in parallel. So, there are Verification phases on one side of the 'V' and Validation phases on the other side. The Coding Phase joins the two sides of the V-Model.

The following illustration depicts the different phases in a V-Model of the SDLC.

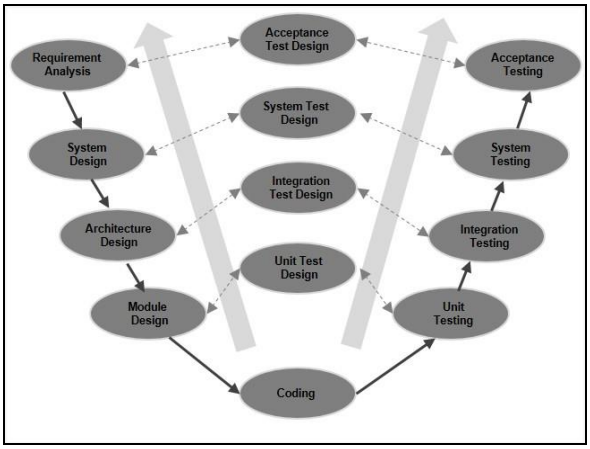

Fig 5. V-Model

Just like the waterfall model, the V-Shaped life cycle is a sequential path of execution of processes. Each phase must be completed before the next phase begins. Testing is emphasized in this model more than the waterfall model. The testing procedures are developed early in the life cycle before any coding is done, during each of the phases preceding implementation. Requirements begin the life cycle model just like the waterfall model. Before development is started, a system test plan is created. The test plan focuses on meeting the functionality specified in requirements gathering. The high-level design phase focuses on system architecture and design. An integration test plan is created in this phase in order to test the pieces of the software systems ability to work together. However, the low-level design phase lies where the actual software components are designed, and unit tests are created in this phase as well. The implementation phase is, again, where all coding takes place. Once coding is complete, the path of execution continues up the right side of the V where the test plans developed earlier are now put to use.

V- Model – Application:

V- Model application is almost the same as the waterfall model, as both the models are of sequential type.

Requirements have to be very clear before the project starts, because it is usually expensive to go back and make changes. This model is used in the medical development field, as it is strictly a disciplined domain.

The following pointers are some of the most suitable scenarios to use the V-Model application.

- Requirements are well defined, clearly documented and fixed.
- Product definition is stable.
- Technology is not dynamic and is well understood by the project team.
- There are no ambiguous or undefined requirements.
- The project is short.

V-Model Analysis:

The advantages of the V-Model method are as follows −

- This is a highly-disciplined model and Phases are completed one at a time.
- Works well for smaller projects where requirements are very well understood.
- Simple and easy to understand and use.
- Easy to manage due to the rigidity of the model. Each phase has specific deliverables and a review process.

The disadvantages of the V-Model method are as follows −

- High risk and uncertainty.
- Not a good model for complex and object-oriented projects.
- Poor model for long and ongoing projects.
- Not suitable for the projects where requirements are at a moderate to high risk of changing.
- Once an application is in the testing stage, it is difficult to go back and change functionality.
- No working software is produced until late during the life cycle.

E. Big Bang Model

The Big Bang Model comprises of focusing all the possible resources in the software development and coding, with very little or no planning. The requirements are understood and implemented as they come. Any changes required may or may not need to revamp the complete software.

Fig 6. Big Bang Model

This model is ideal for small projects with one or two developers working together and is also useful for academic or practice projects. It is an ideal model for the product where requirements are not well understood and the final release date is not given.

Big Bang Model – Analysis:

The advantage of this Big Bang Model is that it is very simple and requires very little or no planning. Easy to manage and no formal procedure are required. However, the Big Bang Model is a very high risk model and changes in the requirements or misunderstood requirements may even lead to complete reversal or scraping of the project. It is ideal for repetitive or small projects with minimum risks.

The advantages of the Big Bang Model are as follows −

- This is a very simple model
- Little or no planning required
- Easy to manage
- Very few resources required
- Gives flexibility to developers
- It is a good learning aid for new comers or students.

The disadvantages of the Big Bang Model are as follows −

- Very High risk and uncertainty.
- Not a good model for complex and object-oriented projects.
- Poor model for long and ongoing projects.
- Can turn out to be very expensive if requirements are misunderstood.

F. Rapid Application Development (RAD) Model

RAD is a linear sequential software development process model that emphasizes a concise development cycle using an element based construction approach. If the requirements are well understood and described, and the project scope is a constraint, the RAD process enables a development team to create a fully functional system within a concise time period.

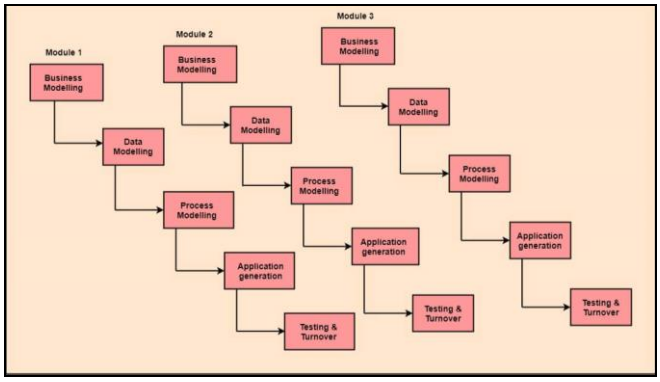

Fig 7. RAD Model

The various phases of RAD are as follows:

- Business Modelling: The information flow among business functions is defined by answering questions like what data drives the business process, what data is generated, who generates it, where does the information go, who process it and so on.
- Data Modelling: The data collected from business modeling is refined into a set of data objects (entities) that are needed to support the business. The attributes (character of each entity) are identified, and the relation between these data objects (entities) is defined.
- Process Modelling: The information object defined in the data modeling phase are transformed to achieve the data flow necessary to implement a business function. Processing descriptions are created for adding, modifying, deleting, or retrieving a data object.
- Application Generation: Automated tools are used to facilitate construction of the software; even they use the 4th GL techniques.
- Testing & Turnover: Many of the programming components have already been tested since RAD emphasis reuse. This reduces the overall testing time. But the new part must be tested, and all interfaces must be fully exercised.

RAD Model – Application:

RAD model can be applied successfully to the projects in which clear modularization is possible. If the project cannot be broken into modules, RAD may fail.

The following pointers describe the typical scenarios where RAD can be used −

- RAD should be used only when a system can be modularized to be delivered in an incremental manner.
- It should be used if there is a high availability of designers for modeling.
- It should be used only if the budget permits use of automated code generating tools.
- RAD SDLC model should be chosen only if domain experts are available with relevant business knowledge.
- Should be used where the requirements change during the project and working prototypes are to be presented to customer in small iterations of 2-3 months.

RAD Model – Analysis:

RAD model enables rapid delivery as it reduces the overall development time due to the reusability of the components and parallel development. RAD works well only if high skilled engineers are available and the customer is also committed to achieve the targeted prototype in the given time frame. If there is commitment lacking on either side the model may fail.

The advantages of the RAD Model are as follows −

- Changing requirements can be accommodated.
- Progress can be measured.
- Iteration time can be short with use of powerful RAD tools.
- Productivity with fewer people in a short time.
- Reduced development time.
- Increases reusability of components.
- Quick initial reviews occur.
- Encourages customer feedback.
- Integration from very beginning solves a lot of integration issues.

The disadvantages of the RAD Model are as follows −

- Dependency on technically strong team members for identifying business requirements.
- Only system that can be modularized can be built using RAD.
- Requires highly skilled developers/designers.
- High dependency on modeling skills.
- Inapplicable to cheaper projects as cost of modeling and automated code generation is very high.
- Management complexity is more.

- Suitable for systems that are component based and scalable.
- Requires user involvement throughout the life cycle.
- Suitable for project requiring shorter development times.

G. Agile Model

The meaning of Agile is swift or versatile. "Agile process model" refers to a software development approach based on iterative development. Agile methods break tasks into smaller iterations, or parts do not directly involve long term planning. The project scope and requirements are laid down at the beginning of the development process. Plans regarding the number of iterations, the duration and the scope of each iteration are clearly defined in advance.

Here is a graphical illustration of the Agile Model –

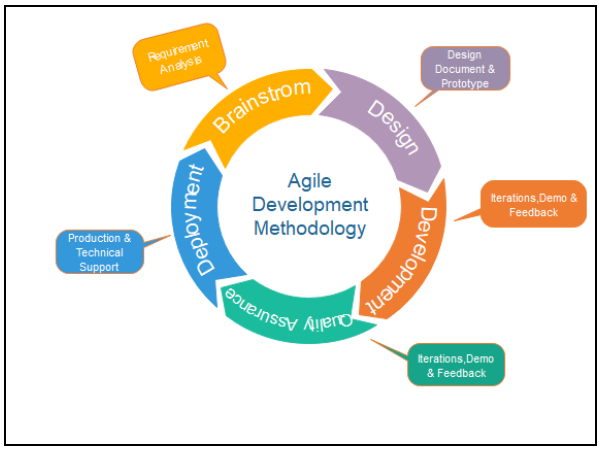

Fig 8. Agile Model

The phases in the Agile Model are as follows:

- Requirements gathering: In this phase, you must define the requirements. You should explain business opportunities and plan the time and effort needed to build the project. Based on this information, you can evaluate technical and economic feasibility.
- Design the requirements: When you have identified the project, work with stakeholders to define requirements. You can use the user flow diagram or the high-level UML diagram to show the work of new features and show how it will apply to your existing system.
- Construction/ iteration: When the team defines the requirements, the work begins. Designers and developers start working on their project, which aims to deploy a working product. The product will undergo various stages of improvement, so it includes simple, minimal functionality.
- Testing: In this phase, the Quality Assurance team examines the product's performance and looks for the bug.
- Deployment: In this phase, the team issues a product for the user's work environment.
- Feedback: After releasing the product, the last step is feedback. In this, the team receives feedback about the product and works through the feedback.

Agile Testing Methods:

- Scrum
- Crystal
- Dynamic Software Development Method(DSDM)
- Feature Driven Development(FDD)
- Lean Software Development
- eXtreme Programming(XP)

When to use the Agile Model?

- When frequent changes are required.
- When a highly qualified and experienced team is available.
- When a customer is ready to have a meeting with a software team all the time.
- When project size is small.

Agile Model – Analysis:

Agile methods are being widely accepted in the software world recently. However, this method may not always be suitable for all products. Here are some pros and cons of the agile model.

The advantages of the Agile Model are as follows −

- It is a very realistic approach to software development.
- Promotes teamwork and cross training.
- Functionality can be developed rapidly and demonstrated.
- Resource requirements are minimum.
- Suitable for fixed or changing requirements
- Delivers early partial working solutions.
- Good model for environments that change steadily.
- Minimal rules, documentation easily employed.
- Enables concurrent development and delivery within an overall planned context.
- Little or no planning required.
- Easy to manage.
- Gives flexibility to developers.

The disadvantages of the Agile Model are as follows −

- Not suitable for handling complex dependencies.
- More risk of sustainability, maintainability and extensibility.
- An overall plan, an agile leader and agile PM practice is a must without which it will not work.
- Strict delivery management dictates the scope, functionality to be delivered, and adjustments to meet the deadlines.
- Depends heavily on customer interaction, so if customer is not clear, team can be driven in the wrong direction.
- There is a very high individual dependency, since there is minimum documentation generated.
- Transfer of technology to new team members may be quite challenging due to lack of documentation.

IV. COMPARATIVE ANALYSIS

Waterfall Model is easy to manage due to the rigidity of the model as each phase has specific deliverables and a review process. It works well for smaller projects where requirements are very well understood.

V-shaped Model has higher chance of success over the waterfall model due to the development of test plans during the life cycle. It works well for small projects where requirements are easily understood.

Iterative model is at the heart of a cyclic software development process. It starts with an initial planning and ends with deployment with the cyclic interactions in between. Easier to test and debug during a smaller iteration. Easier to manage risk because risky pieces are identified and handled during its iteration.

Spiral model is good for large and mission-critical projects where high amount of risk analysis is required like launching of satellites.

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.

It can be conferred that Agile SDLC excels traditional SDLC.

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

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. 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. 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. 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. 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. 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. One immediate consequence of these dependency issues is the violation of the Interface Segregation Principle, explaining most of the difficulties in the deployment stage

V. 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 are 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.

REFERENCES

[1] Ian Sommerville, "Software Engineering", Addison Wesley, 7th edition, 2004.

[2] Steve Easterbrook, "Software Lifecycles", University of Toronto Department of Computer Science, 2001.

[3] National Instruments Corporation, "Lifecycle Models", 2006 , http://zone.ni.com.

[4] JJ Kuhl, "Project Lifecycle Models: How They Differ and When to Use Them",2002 www.business-esolutions.com.

[5] Karlm, "Software Lifecycle Models', KTH,2006 .

[6] Rlewallen, "Software Development Life Cycle Models", 2005, http://codebeter.com.

[7] Barry Boehm, "Spiral Development: Experience, Principles, and Refinements", edited by Wilfred J. Hansen, 2000.

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

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

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

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

[12] 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.

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

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

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

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