Comparative Analysis on Software Development Life Cycle (SDLC) Models

G.A Monang Lumban Gaol[1] and Setyawan Widyarto[2]

Program Studi Magister Ilmu Komputer, Universitas Budi Luhur
Jl. Ciledug Raya, RT.10/RW.2, Petukangan Utara, Kec. Pesanggrahan, Kota Jakarta Selatan, Daerah Khusus Ibukota Jakarta 12260

[1]2111600371@student.budiluhur.ac.id
[2]swidyarto@gmail.com

***Abstract* - There are some different software development models that are widely accepted as a software development lifecycle model. Selecting the right life cycle model is the most valuable process while a developer has to complete within a given time deadline and estimated cost. Model selection depends on the type of project as per customer requirement. Although all these models have their own benefits and drawbacks, the combination of all these methodologies is adopted in the existing commercial software development lifecycles.**

***Keywords* — Combining Technique, Comparative Analysis, SDLC, SDL Models, Software Development.**

I. INTRODUCTION

There are four journals that have been reviewed by taking the topic of comparative analysis between software development life cycle models with the aim of providing a choice of models that are suitable for use by software developers according to the project to be worked on in such a way that software development can run more effectively and in accordance with wants or needs of system users. There are various kinds of software development models used from the four reviewed journals which generally use models that are quite popular for software developers, such as waterfall, ESD, agile methodology, spiral model, component based, etc.

In performing comparative analysis, among the authors, some performed through direct comparisons and others performed comparisons after performing the combining technique.

From the comparative analysis that has been done, the authors generally conclude that the selection model depends on the type of project as per customer requirement. And in essence, selecting the correct life cycle model is the most valuable process while the developer has to complete within a given time deadline and estimated cost.

II. METHOD

A. Comparative Analysis by Aminu and Ogwueleka

This paper attempts to discuss the different available Software Development Lifecycle (SDLC) models and the scenarios in which these models are used. All popular approaches to SDLC are being discussed, both structured, object-oriented, and agile methods. The paper also highlights the benefits and drawbacks of the models discussed as well as the practical applications. This will help project managers decide what SDLC model would suit their project and also help developers and testers understand the basics of the development model being used for their project.

Waterfall model is simple and easy to use and understand, because of the rigidity of the model, easy to manage. Each phase has specific results and a review process, that is one phase at a time is processed and completed. Works well for smaller projects where there is a very good understanding of requirements and the process are well documented. The main drawback of waterfall model is that the design has to be completely specified before programming begins and it takes a long time from the completion of the system proposed in the analysis phase and the delivery of the system. Therefore, it is not suitable for the complex project because it cannot accommodate the changing requirement; adjusting scope during the life cycle can end a project.

In Iterative Model, some functionality in the life cycle can be developed quickly and early. Results are fast and frequently obtained. During iteration, risks are identified and resolved; and each iteration milestone is easily managed. Progress is measured, while the operational product is delivered with each increment. Issues, challenges & risks identified from each increment can be used/applied to the next increment. It supports changing requirements. Software is produced early during the life cycle, which facilitates customer evaluation and feedback. Better suited to large and mission-critical projects. It is the main drawback is it needs more attention to management, the management complexity is more. The progress of the project is highly dependent on the phase of risk analysis. Highly skilled resources are required for risk analysis. Not suitable for smaller projects.

The spiral model accommodates changing requirements, and users have an early view of the system. Same as in the iterative model, the management is more complex and not suitable for small projects. A very complex process, the spiral may go on indefinitely with require also required excessive documentation for a large number of intermediate stages. It is not possible to know the end of the project early. The spiral model has been very influential in helping people think about iteration in software processes and introducing the risk-driven approach to development. In practice, however, the model is rarely used.

V-model works well for smaller projects where there is a very good understanding of requirements. Simple and easy to use and understand, because of the rigidity of the model, easy to manage–each phase has specific results and a review process. It is a disadvantage is a High risk and uncertainty. Once an application has been tested, it's hard to go back and change the functionality. This shows that it is not a good model for projects that are complex and object-oriented.

Similarly, Big band model is a very simple model, which requires little or no planning and very few resources. Gives flexibility to developers which serve as a good learning aid to newcomers or students. Same as with Vmodel, it is not a good model for projects that are complex and object-oriented and can be very expensive.

Furthermore, the Agile model is easy to use, provides flexibility to developers. It promotes cross-training and teamwork where functionality can be developed rapidly and demonstrated. The requirements for resources are minimal, very suitable for requirements fixed or changing that delivers limited early work solutions. It is a good model for ever-changing environments. It requires an overall plan, an agile leader, and Agile Project Management (PM) practice without which it won't work. More risk of sustainability, maintainability, and extensibility. This shows it is not suitable for the handling of complex dependencies. Individual dependency is very high since minimal documentation is generated. Technology transfer to new team members may be quite challenging due to lack of documentation.

Additionally, In RAD Model, Model Changing requirements may be accommodated. It is possible to measure progress as in the Iterative model. It has minimal time for development and increases component reusability. It depends on high modeling skills and technically strong team members to identify business requirements, which required highly qualified developers /designers. This shows that it is not applicable as a cost to cheaper projects.

Finally, the Prototype Model increased user involvement compare to other models; the users will have a better understanding of the system that is being developed because a functioning model of the system is displayed. Reduces time and cost as it is possible to detect defects much earlier. Quicker user feedback for better solutions is available by identifying missing and confusing functions. There is a risk of insufficient requirement analysis due to too much prototype dependence. In practice, this methodology can increase the system's complexity as the system's scope can extend beyond the original plans. The effort invested in the building of prototypes could be excessive. So, a prototype is useful when a customer or developer is not sure of the requirements, or of algorithms, efficiency, business rules, response time, etc. Prototyping is not a standalone, complete development methodology, but rather an approach to be used in the context of a full methodology (such as incremental, spiral, etc.). Incremental software development is better than a waterfall approach for most business, e-commerce, and personal systems. By developing the software incrementally, it is cheaper and easier to make changes in the software as it is being developed. Compared to the waterfall model, incremental development has some important benefits such as the cost of accommodating changing customer requirements is reduced as the amount of analysis and documentation that has to be redone is much less than what is required with the waterfall model; It is easier to get customer feedback on the work done during development than when the system is fully developed, tested, and delivered; and more rapid delivery of useful software is possible even if all the functionality has not been included. Customers can use and gain value from the software earlier than it is possible with the waterfall model.

B. Combining Teschnique Analysis by Pradhan

Combined technique towards the event innovations of a brand new software package style Life cycle considering numerous existing model specifications, their constraints and limits.

While there are several reasons to use organic process development on a project, that specialize in one or 2 crucial edges can facilitate optimize efforts. These goals can guide later selections like a way to structure user involvement, a way to amendment plans. In response to user feedback, and the way to prepare the project. Notwithstanding what goals are targeted on, it's crucial to speak the explanations for strategic selections to each management and therefore the development team. Evolutionary development may be a completely different approach of puzzling over managing computer code comes. Most teams can in all probability expertise a number of the pain that typically accompanies amendment; therefore it's better to start out with a little trial 1stAnd then attempt a bigger project.

C. Comparative Analysis by Chopra and Nautiyal

This paper tells the efficiency of various CBSE models which are useful for the software project.

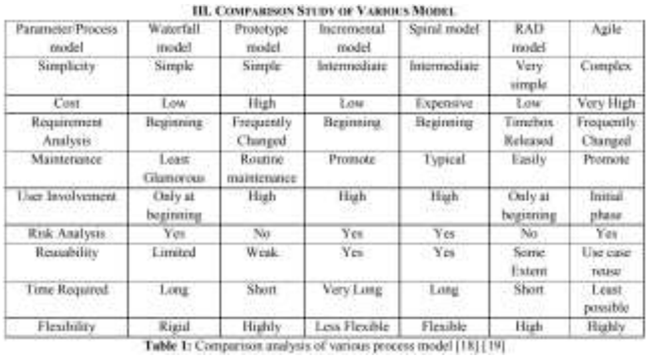
CBSD approach is based on the idea to develop software systems by selecting appropriate off-the-shelf components and then to assemble them with a welldefined software architecture. The purpose of CBSD is to develop large systems, incorporating previously developed or existing components, thus cutting down on development time and costs.

CBSE can also be used to reduce maintenance associated with the upgrading of large systems. It is assumed that common parts in a software application only need to be written once and reused rather than being rewritten every time a new application is developed.Component primarily based software package development approach relies on the thought to develop software package systems by choosing acceptable ready to wear components and so to assemble them with a well-defined package design.

CBSE encompasses two parallel engineering activities, domain engineering and component-based development (CBD). Domain engineering explores the application domain with the specific intent of finding functional, behavioral, and data components that are candidates for reuse and places them in reuse libraries. CBD elicits requirements from the customer and selects an appropriate architectural style to meet the objectives of the system to be built. This new software development approach is very different from the traditional approach in which software systems can only be implemented from scratch. These commercial off-the shelf (COTS) components can be developed by different developers using different languages and different platforms.

D. Camparison Analysis by Polishwala and Shastri

The aim of this paper is to analysis some methodologies that gives comparison on SDLC models by studied available, tools, techniques and methodologies of SDLC models.



III. COMPARISON STUDY OF VARIOUS MODEL

Parameter/Process model	Waterfall model	Prototype model	Incremental model	Spiral model	RAD model	Agile
Simplicity	Simple	Simple	Intermediate	Intermediate	Very simple	Complex
Cost	Low	High	Low	Expensive	Low	Very High
Requirement Analysis	Beginning	Frequently Changed	Beginning	Beginning	Timebox Released	Frequently Changed
Maintenance	Least Glamorous	Routine maintenance	Promote	Typical	Easily	Promote
User Involvement	Only at beginning	High	High	High	Only at beginning	Initial phase
Risk Analysis	Yes	No	Yes	Yes	No	Yes
Reusability	Limited	Weak	Yes	Yes	Some Extent	Use case reuse
Time Required	Long	Short	Very Long	Long	Short	Least possible
Flexibility	Rigid	Highly	Less Flexible	Flexible	High	Highly

Table 1: Comparison analysis of various process model [18] [19]

III. CONCLUSION

Aminu and Ogwueleka conclude that model selection depends on the type of project as per customer requirement. Though these models all have their benefits and drawbacks, the fusion of all these methodologies is incorporated in the existing commercial software development world.

Pradhan said that the most salient and consistent edges of the ESD model are its ability to induce early, accurate well shaped feedback from users and therefore the ability to reply thereto feedback.

Chopra and Nautiyal concluded that the key factor was based on the reasoning of the researcher. On the basis of analyzing is that Knot Model is the best model for the software project.

Polishwala and Shastri conclude that there are many SDLC models such as waterfall, prototype, spiral, incremental, RAD, Agile etc. used in various organizations. Each model has its own pros and cons. In software industry, all this methodologies are used, so here the comparison of all these models is provided basis of certain features like – requirement specification, cost, simplicity expertise, risk involvement, flexibility, maintenance, etc. from comparison through these basic features helps developers to select appropriate model for particular project. Selecting the correct life cycle model is most valuable process while developer has to complete within a given time deadline and estimated cost. This study make SDLC selection process easy and effective for the system.

REFERENSI

[1] Aminu, Halima, & Ogwueleka, F. (2020). A Comparative study of System Development Life Cycle Models.

[2] Pradhan, Debasis, Dalai, Sasank Sekhar, & Behera, Mandakini Priyadarsini (2020). A Comparative Study on Evolutionary Model for Software Development

[3] Chopra, Mr. Sandeep, Sharma, M., & Nautiyal, Lata (2017). Comparative Study of Different Models in Component Based Software Engineering

[4] Polishwala, Megha V., & Shastri, Dr. Amit kumar (2021). Comparative Analysis of Various Software Development