A Comparative Study of System Development Life Cycle Models

[1]Halima Aminu

Department of Computer Science, Nigerian Defence Academy.

[2]Francisca Nonyelum Ogwueleka

Department of Computer Science, Nigerian Defence Academy.

Abstract

Over the decades, software development models have morphed with the choice of the methodology being influenced by several factors such as clarity of user requirements; familiarity with base technology; system robustness; the need for system reliability; time pressure; and need to see improvements on schedule. System Development Life Cycle (SDLC) are abstractions of the process that can be used to explain different approaches to software development. They can be adapted and extended to create more specific processes. Almost all software developers faced challenged in choosing the right software development process. There are various models of SDLC with different characteristics, advantages, and drawbacks. The success of a software project largely depends on which system design is used for development, as information technology projects and solutions have become extremely important in institution/organizations of all sizes. In this paper, we make a comparative analysis of different SDLC models; we further discuss their benefits and their drawbacks, which help software developers/designers to choose the right models for the software development project.

Keywords: *System Development Life Cycle (SDLC), SDLC Models, waterfall model, iterative model, prototype model*

1.0 Introduction

For the last three decades, the disruptive innovation in Information Technology (IT) has changed almost all the human endeavor. There are hardly any areas where IT are not being used. As a result of this development, individuals, institutions, and organizations rely heavily on software. Pressman (2005) argued that software became a decision-making tool in the contemporary world of business, academic research, and implementing government policies. Similarly, software and software-related models in the form of information systems has simplified the services provided by businesses and government in areas such as agriculture, entertainment, manufacturing, medicine, telecommunications, and transport sector. At the same time, this software's are developed based on users need; therefore, the developers need to have an in-depth knowledge of the problem the software need to solve. This necessitates the software development life cycle where the active partners (institution/organization and the developers) design a development framework that will guide them throughout the software development phase.

The SDLC is an application development life cycle as used in system and software engineering. SDLC is a distinct structure that composed of various phases which are used by the system engineers and software developers for creating, planning, implementing, testing, deploying of the software product and up to the system maintenance period (Ragunath, 2010).

Different forms of software development models that currently exist in building a framework or framework are based on the concept of SDLC. According to Rainer (2009), SDLC is a structured framework that contains various processes in which the system is developed. The SDLC process comprises of the hardware, software or the combination of both which they aim to produce a high-quality system that meets the requirement of customer/ user or even exceeds their expectations within stipulated time frames and cost estimates (Sommerville, 2010).

Computer Systems are extremely complex, to manage this level of complexity. Several SDLC models have been created such as Waterfall Petersen (2009), Iterative Jalote (2004), Incremental Lochau (2012), Spiral Ruparelia (2010), V-model Balaji (2012), Big band Bhuvaneswari (2013), Agile Stoica (2013), RAD Kumar (2014), and Prototype models Isaias (2015). Development of good software solution requires a proper process to be followed to estimate as well to predict the reliability of software systems. This software process which is required to produce software differs. A proper software life cycle model provides a set of activities to be carried out during system development. Software development is divided into a set of activities that allow any software development company to control the software product easily. SDLC use step by step approach to complete the software development process. The software development cycle is all about, problem domain (understanding the problem), solution domain (planning for the solution), designing and coding of planned solution, testing the actual program, and maintenance of the product.

As a systematic approach, SDLC performs multi-stage software development. There are different life cycle models for software development that are used in software design processes; each has its benefits and drawbacks, as illustrated in Table 1. The software developers face a lot of challenges during the process of software development including; choosing the right model which will enable the development strategies to be set in identifying the best model that fits the goals of the software to be designed. To end this, we perform a comparative analysis of these models, which would guide and aid the challenges of the software development process.

Table 1: Comparison of Structured, Object-oriented and Agile/adaptive methods

	SDLC MODELS									
Ability to Develop Systems	**WATERFALL**	**ITERATIVE**	**PARALLEL**	**PROTOTYPE**	**PHASED**	**THROWAWAY PROTOTYPE**	**SPIRAL**	**BUILD & FIX**	**V-SHAPED**	**AGILE**
With Unclear User Requirements	Poor	Good	Poor	Excellent	Good	Excellent	Good	Good	Poor	Excellent
With Unfamiliar Technology	Poor	Good	Poor	Poor	Good	Excellent	Good	Good	Poor	Poor
that is Complex	Good	Good	Good	Poor	Good	Excellent	Poor	Poor	Good	Poor
that are reliable	Good	Good	Good	Poor	Good	Excellent	Good	Good	Excellent	Good
with Short Time Schedule	Poor	Excellent	Good	Excellent	Excellent	Good	Poor	Poor	Poor	Excellent
with Schedule Visibility	Poor	Excellent	Poor	Excellent	Excellent	Good	Excellent	Poor	Poor	Good

2.0 Related works

Several studies on SDLC models have been performed. Dhami (2016) compared eight SDLC models, namely: Waterfall, Iterative Waterfall, Prototype, Rapid Application Development Model (RAD), Incremental, Build and Fix, Spiral, and V-shaped model. Sushma (2017) made a comparison of three models, namely waterfall, iterative, and prototyping. After their analysis, they find out that various large companies use the waterfall model for their projects. Even though the development team is familiar with the environment, and it is

feasible to specify all work environment requirements. Iterative enhancement model overcomes the waterfall model's drawback. It allows feedback to proceed with the phase. Prototype model used to develop transaction processing online systems. Since it significantly reduces rework and leads to the creation of a working model at lower capital cost. Davis (1988), provided a framework which serves as a basis for analyzing the similarities and differences among alternate life-cycle models. Sharma (2009), performed a comparative analysis of software process improvement models their importance and their drawbacks.

Maglyas (2010), described that the size and complexity of software development projects are growing; at the same time, the proportion of successful projects is still quite low. Rothay (1989), provided a brief review of traditional SDLCs, they related how the use of traditional software development models is numerous and often regarded as the proper and disciplined approach to the analysis and design of software applications. Osborn (2001), discussed traditional SDLC techniques and how over time the phases of these approaches have become enshrined in a development cycle that includes defining requirements, designing a system to meet those requirements, coding, and testing.

3.0 System Development Life Cycle

According to Rainer (2014), SDLC is a structured framework consisting of sequential processes through which information systems are developed. The software life cycle is the period that starts when a software product is conceived and ends when the product is no longer available for use. The software life cycle typically includes the following activities which includes; Systems investigation, system analysis, system design, programming and testing, implementation, operation, and maintenance as depicted in Figure 1. These activities above form part of SDLC framework activities and are performed in every software development project.

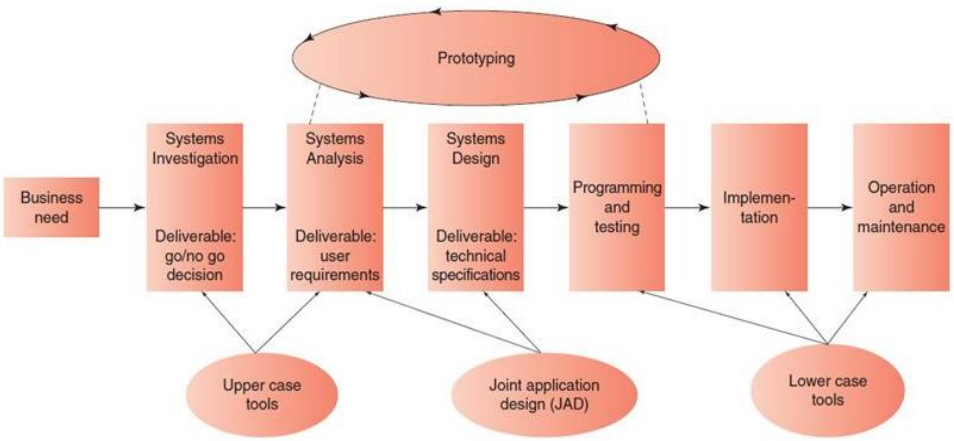


Figure 1: SDLC Stages with supporting Tools (Rainer, 2014)

Each of the outlined software development processes mentioned are further broken down into several stages, allowing a software development company to efficiently manage the work of building software products from the required functionality within a specific timeframe and budget. All software projects go through the stages of requirements collection, business analysis, system design, implementation, and quality assurance testing (Klopper, 2007).

3.1 System Investigation

The first stage in software development in traditional SDLC is system investigation. Software's developers agree that the more time they invest in understanding the user's needs, the problem to be solved, the technical details of the systems and the anticipating problems to encounter they more likely to succeed during the development phase. For these reasons, system investigation using the feasibility study addresses the business problem or a business opportunity (Rainer, 2014). The top management decides from the feasibility analysis, whether to proceed with the project or stop it. If the decision is to go, the project will proceed, and the analysis phase of the systems will begin.

3.2 Systems Analysis

Systems analysis is the process by which system analysts examine the problems that the institution or organization plans to solve with an information system. The primary purpose of the analysis stage of the systems is to collect information about the existing system if it is available to determine how to enhance it if no system exists the information obtained in the first stage are extensible review to develop a new system (Rainer, 2014). The next step is system design after accumulating the user requirement for the new system.

3.3 Systems Design

System design describes how the system will solve the institution/organization problem. When all participants approve the system specifications, those specifications are frozen, that should not be changed. Adding functions after initiation of the project causes the scope to shrink, in which the time frame and costs associated with the project extend beyond the agreed limits. Scope creep puts both the budget of the project and its schedule at risk. Due to the expensive scope creep, successful project managers place controls on user-requested changes. These controls help prevent bestselling projects.

3.4 Programming and Testing

This phase involves translating the design specifications into computer code. Depending on the complexity of the project, the code can range from a few lines to thousands of codes. These projects involve forming teams in programming. The teams often include users in the functional area who help programmers focus on the business issue. During the programming stage, thorough and continuous testing takes place. Testing is the process that evaluates whether the computer code produces the expected results. It is also intended to detect errors and bugs which need to be debugged before final deployment.

3.5 Implementation

Implementation (deployment) is the process of converting to a new one from an old computer system. Three major conversion strategies are used by organizations: direct, pilot, and phased. In direct conversion, the new system is being mounted in the whole organization while the old system is withdrawn entirely. This is the least expensive of all the three. It is also the riskiest because, if the new system does not work as planned, there is no support from the old system. Because of these risks, few systems are implemented using direct conversion.

Furthermore, in pilot conversion, deploy the new system in one plant or one functional area. For some time, the new system runs and is then evaluated. If the evaluation confirms the proper functioning of the system, the system will then be implemented in other parts of the organization. A phased conversion introduces stages of the new system components, such as individual modules. Every module is evaluated. If it works properly, other modules will be introduced until the whole new system is operational.

3.6 Operation and Maintenance

It will operate for some time after the new system is implemented until it no longer meets its goals. Once the operations of the new system are stabilized, the company conducts an audit to evaluate the capabilities of the system and to determine whether it is properly used. Systems require several types of maintenance. The first type is program debugging, a process that goes on throughout the system life. The second type is to update the system to accommodate changes in the conditions of business. Usually, these corrections and upgrades do not add any new features. Instead, they assist the system in achieving its goals.

4.0 System Development Lifecycle Model Classifications

There are three popular approaches to system development, which includes; 1) Structured analysis, 2) object-oriented analysis (O-O), and 3) Agile methods.

4.1 Structured Analysis

According to Shelly (2012), structured systems analysis is a traditional method of systems development that is time-tested and easy to understand, for the construction of a building, the structured analysis uses an overall plan, similar to a blueprint. This type of method makes use of process models to describe a system graphically and also addresses the problems with data organization and structure, relational database design, and user interface issues.

4.2 Object-oriented Analysis

Technically speaking, object-oriented approaches to the development of information systems can use any of the traditional methodologies (development of waterfalls, parallel development, V-model, iterative development, prototyping of systems, and throwaway prototyping). The object-oriented approaches are mostly associated with the Rapid Application Development (RAD) methodology for iterative development. However, the main difference between a traditional approach, such as structured design and an object-oriented approach is how to decompose it. The problem of decomposition is either process-centered or data-centered in traditional approaches. The object-oriented analysis combines data and processes, acting on the data into things called objects, which they exhibit some characteristics called properties. A programmer then writes the code that creates the objects using an O-O programming language. Object-oriented methods typically follow a series of SDLC-like phases of analysis and design, but the phases are more interactive.

4.3 Agile Methods

This is the newest approach that tries to progressively develop a system by streamlining the SDLC and eliminating any of the tasks and time associated with the definition and documentation of requirements. Agile methods usually use a spiral model based on user feedback that represents a series of iterations or

modifications. The repeated iterations result in a series of prototypes evolving into the finished system. Regardless of the development strategy, using project management tools and techniques, people, tasks, timetables, and costs must be managed effectively.

5.0 Software Development Process Models

Process models are used to explain how processes relate to system users, how processes relate to each other, how processes enter and generate data, and how processes create and use data stored. However, the process includes the process description, which includes, Products (that is the outcomes of the activity, such as the outcome of an architectural design model for the software architecture), Roles (this is the responsibilities of the people involved in the process, such as the project manager and the programmer), Pre and post-conditions (These are conditions that must be true before and after an activity). The precondition of the architectural design is the requirements that have been approved by the customer, while the post condition is the diagrams describing the architectural have been reviewed.

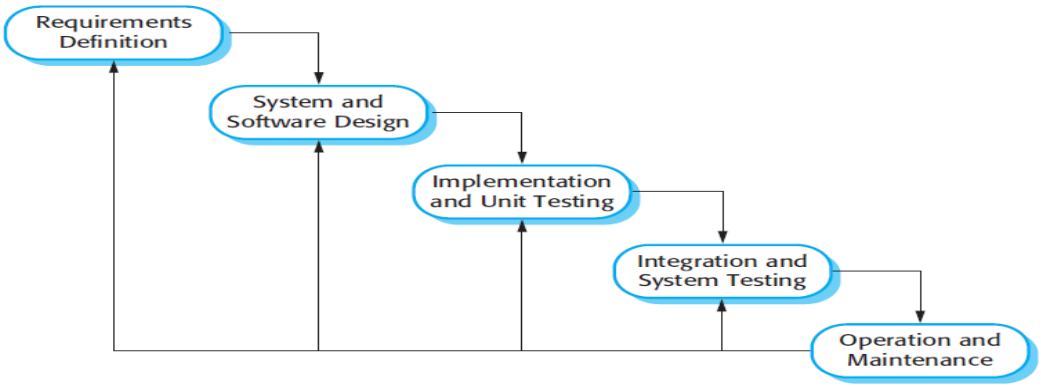
A software process model is a simplified representation of a software process. Each model represents a process from a specific perspective. The software process is complex; it relies on making decisions. There's no ideal process, and most organizations have developed their software process, because an organization works on critical systems has a very structural process, while with business systems, with rapidly changing requirements, a less formal, flexible process is likely to be more effective. Process models help to clarify the software components required to fulfill the functional requirements.

Furthermore, the functional requirements begin to define the data to be tracked to accomplish user tasks (Dennis, 2012). The various SDLC models are as follows;

5.1 Waterfall Model

In this approach, the entire software development process is divided into separate phases where the result of one phase sequentially acts as the input for the next phase, as illustrated in Figure 2. Each phase's result is called a deliverable or end product that flows into the next phase. In the waterfall model, you must plan and schedule all of the activities before starting working on them (plan-driven process). The plan-driven process is a process where all the activities are planned first, and the progress is measured against the plan. While the agile process, planning is incremental, and it's easier to change the process to reflect requirement changes. The phases of the waterfall model are requirements, design, implementation, testing, and maintenance.

Figure 2. Structure of Waterfall Model (Sommerville, 2011)



5.2 Iterative Model

This model begins with a simple implementation of a small set of software requirements and improves the evolving versions iteratively until the complete system is implemented and ready for deployment. An iterative model of the life cycle does not attempt to begin with a full requirement specification. Instead, development begins with only part of the software specified and implemented, which is then reviewed to identify additional requirements, as shown in Figure 3. This process is then repeated, and at the end of each model iteration, a new version of the software is produced.

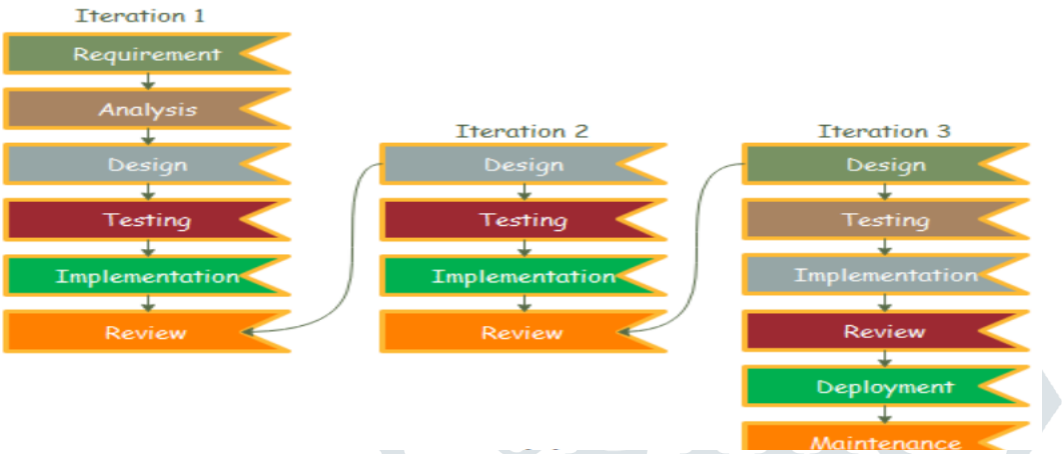


Figure 3. Structure of Iterative Model (javatpoint.com)

5.3 Incremental Model

Incremental model as depicted in Figure 4 is based on the idea of developing an initial implementation, exposing this to user feedback, and evolving it through several versions until an acceptable system has been developed. The activities of a process are not separated but interleaved with feedback involved across those activities. Each system increment reflects a piece of the functionality that is needed by the customer. Generally, the early increments of the system should include the most important or most urgently required functionality. This means that the customer can evaluate the system at an early stage in the development to see if it delivers what's required. If not, then only the current increment has to be changed and, possibly, new functionality defined for later increments.

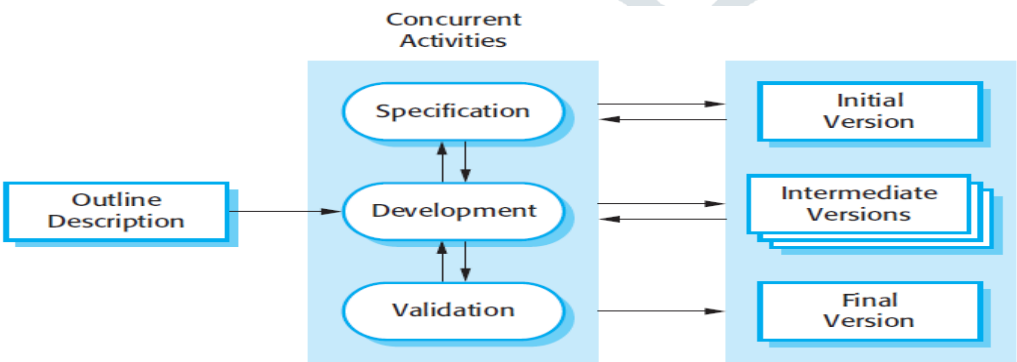


Figure 4. Structure of Incremental Model (Sommerville, 2011)

5.3 Spiral Model

Spiral model is an evolutionary software process model that couples the iterative nature of prototyping with the controlled and systematic aspects of the waterfall model (Ragunath, 2010). It allows incremental product releases or incremental refinement through each iteration around the spiral, as illustrated in Figure 5. The spiral model is a risk-driven where the process is represented as a spiral rather than a sequence of activities. Each loop as shown in Figure 5 in the spiral represents a phase. It initially starts with system feasibility, then requirements definition, then the system design, and so on. Each loop in the spiral is split into four sectors namely: In objective setting, the objectives and risks for that phase of the project are defined, risk assessment and reduction; for each of the identified project risks, a detailed analysis is conducted, and steps are taken to reduce the risk. In development and validation, after risk evaluation, a process model for the system is chosen. So if the risk is expected in the user interface, then we must prototype the user interface. If the risk is in the development process itself, then use the waterfall model. During planning, the project is reviewed, and a decision is made whether to continue with a further loop or not.

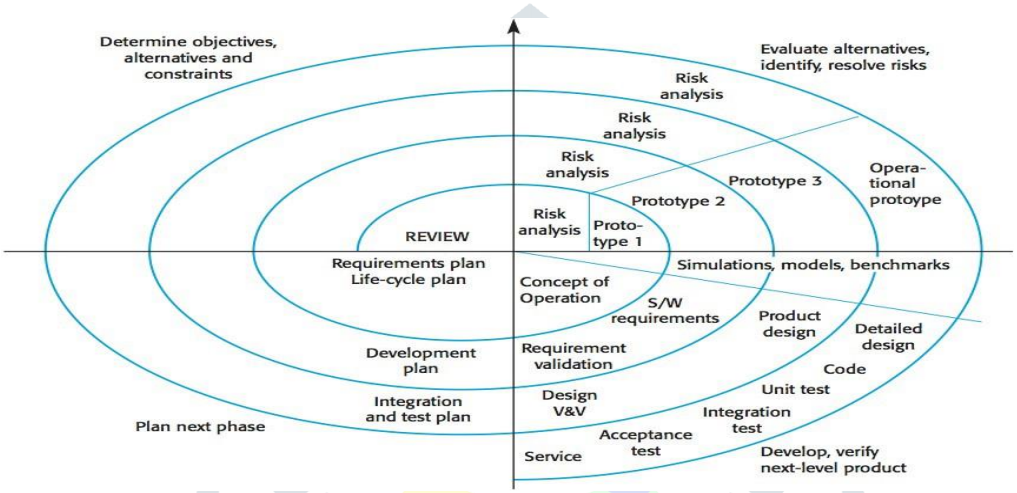


Figure 5. Structure of Spiral Model (Sommerville, 2011)

5.4 V -Model

The V-model is an SDLC model where process execution in V-shape occurs sequentially. It is also referred to as the model of Verification and Validation. V –Model, as shown in Figure 6 is an extension of the waterfall model and is based on the combination of a test phase for each development phase. This means that there is a directly associated test phase for every single phase of the development cycle. This is a highly disciplined model, and the next phase only begins after the previous phase has been completed.

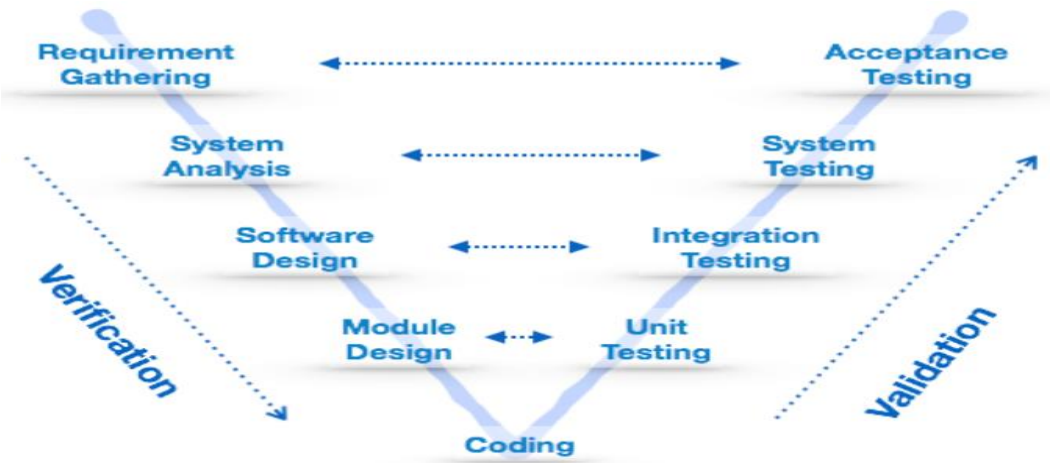


Figure 6. Structure of V- Model (tutoriaspoint.com)

5.6 Big Bang Model

In Big Bang, as illustrated in Figure 7, the model is not followed by any specific process. Development starts with the required money and effort as the input, and the output is the software developed. Big Bang model does not follow any formal development and requires very little planning. This model is usually followed for small projects where the development teams are very small.

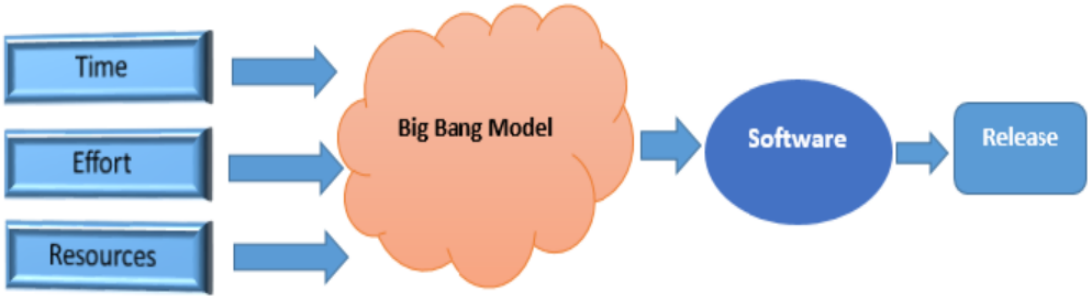


Figure 7. Structure of Big Bang Model (Sommerville, 2011)

5.7 Agile Model

Agile Software Development Lifecycle (SDLC) model is a combination of iterative and incremental process models focusing on process adaptability and customer satisfaction through fast delivery of working software product. As represented in Figure8 Agile models divide the product into small incremental structures. These builds are provided in iterations. Typically, each iteration lasts about one to three weeks. Each iteration involves cross-functional teams working simultaneously on different areas such as planning, analysis of requirements, design, coding, testing of units, and testing of acceptance. At the end of the iteration, the customer and important stakeholders will be shown a working product (Shelly, 2012).

Figure 7. Structure of Agile Model (Sommerville, 2011)

5.8 Rapid Application Development (RAD)

According to Shelly (2012), rapid application development (RAD) as represented in Figure 8 uses a group approach but goes much further. RAD focuses on gathering customer requirements through workshops or focus groups, early customer prototype testing using iterative concept, reuse of existing prototypes (components), ongoing integration, and fast delivery.

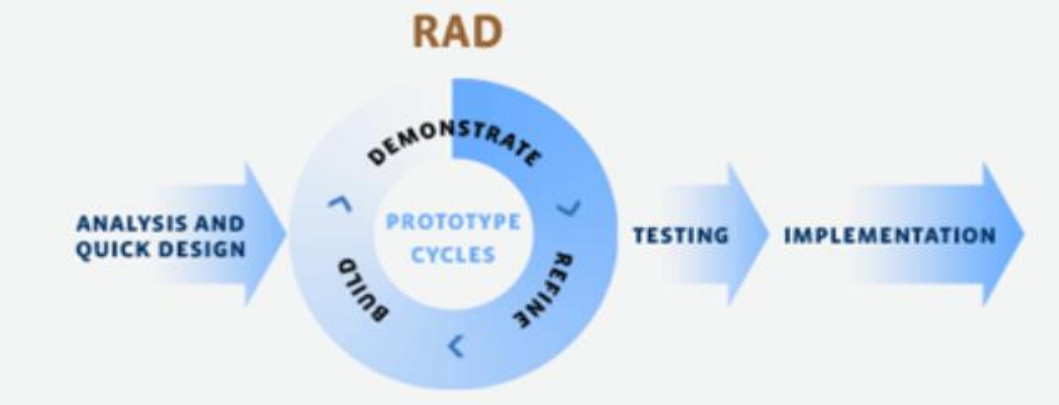


Figure 8. Structure of RAD Model (Sommerville, 2011)

5.9 Prototyping

In prototyping as illustrated in Figure 9., software application prototypes are being developed to depict the product's functionality. As a software development model, software prototyping is becoming very popular as it allows customer needs to be understood at an early stage of development. It helps the customer receive valuable feedback and helps software designers and developers understand what is expected from the product being developed.

In prototyping, the client is involved throughout the development process, which increases the likelihood of client acceptance of the final implementation. While some prototypes are developed with the expectation that they will be discarded, it is possible in some cases to evolve from prototype to a working system.

A software prototype can be used in the requirements engineering, a prototype can help with the elicitation and validation of system requirements. It allows the users to experiment with the system, and so, refine the requirements. They may get new ideas for requirements, and find areas of strength and weakness in the software. Furthermore, as the prototype is developed, it may reveal errors and in the requirements. The specification may be then modified to reflect the changes.

In the system design, a prototype can help to carry out design experiments to check the feasibility of a proposed design. The phases of a prototype are establishing objectives (the objectives of the prototype should be made explicit from the start of the process); define prototype functionality (decide what the inputs and the expected output from a prototype are. To reduce the prototyping costs and accelerate the delivery schedule, you may ignore some functionality, such as response time and memory utilization unless they are relevant to the objective of the prototype); develop the prototype (the initial prototype is developed that includes only user interfaces) and evaluate the prototype (once the users are trained to use the prototype, they then discover requirements errors). With the feedback, both the specifications and the prototype can be improved. If changes are introduced, then a repeat of steps 3 and 4 may be needed.

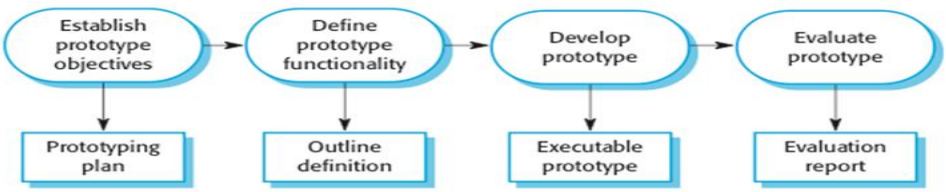


Figure 8. Structure of Prototyping Model (Sommerville, 2011)

6.0 Discussion

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 V-model, 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.

Conclusion

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.

In conclusion, 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.

References

Balaji, S., & Murugaiyan, M. S. (2012). waterfall vs V-Model vs Agile: A comparative Study on SDLC. *International Journal of Information Technology and Business Management*, 2(1) 26-30.

Bhuvaneswari, T., & Prabaharan, S. (2013). A Survey on Software Development Life Cycle Models. *International Journal of Computer Science and Mobile Computing*, 2(5) 262-267.

Davis, A. M., Bersoff, H., & Comer, E. R. (1988). A Strategy for Comparing Alternative Software Development Life Cycle Models. *Journal IEEE Transactions on Software Engineering*, 14(10) 1453-1461.

Dennis, A., Wixom, B. H., & Roth, R. M. (2012). *System Analysis and Design.* New Jersey: John Wiley & Sons, Inc.

Dhami, H. P. (2016). Comparative Study and Analysis of Software Process Models on Various Merits. *International Journal of Advanced Research in Computer Science and Software Engineering*, 6(9) 234-243.

Isaias, P., & Issa, T. (2015). Information System Development Life Cycle Models. *High Level Models and Methodologies for Information Systems. Springer, New York,NY*, 21-40.

Jalote, P., & Palit, A. (2004). Timeboxing: A Process Model for Iterative Software Development. *Journal of Systems and Software. Elsevier*, 117-127.

Klopper, R., Gruner, S., & Kourie, D. G. (2007). Assessment of a Framework to Compare Software Development Methodologies. *South African Institute of Computer Scientists and Information Technologists of Information Technologists on IT Research in Developing Countries*, (pp. 56-65). Port Elizabeth. South Africa.

Kumar, G., & Bhatia, P. K. (2014). Comparative Analysis of Software Engineering Models from Traditional to Modern Methodologies. *IEEE Fourth International Conference on Advanced Computing & Communication Technologies*, (pp. 189-196). Rohtak, Haryana, India.

Lochau, M., Schaefer, I., Kamischke, J., & Lity, S. (2012). Increamental Model-Based Testing of Delta-Oriented Software Product Lines. *Internation Conference on Tests and Proofs.* (pp. 67-82). Berlin, Heidelberg: Springer.

Maglyas, A., Nikula, U., & Smolander, K. (2010). Comparison of Two Models of Success Prediction in Software Development Projects. *Software Engineering Conference* (pp. 43-49). Moscow, Russia: IEEE.

Malik, S., & Nigam, C. (2017). A Comparative Study of Different Types of Models in Software Development Life Cycle. *International Research Journal of Engineering and Technology*, 4(10) 437-440.

Osborn, C. (1995). *SDLC, JAD, and RAD: Finding the Right Hammer.* Retrieved from Center for Imformation Management Studies: Retrieved October 13, 2018, from the World Wide Web: http://faculty.babson.edu/osborn/cims/rad.htm#SDLCvsJADvsRAD

Petersen, K., Wohlin, C., & Baca, D. (2009). The Waterfall Model in Large-Scale Development. *International Conference on Product-Focused Software Process Imporvement* (pp. 386-400). Berlin, Heidelberg: Springer.

Pressman, R. S. (2005). *Software Engineering: A Practitioners Approach.* NY: McGraw-Hill Higher Education.

Ragunath, P., Velmourougan, S., Davachelvan, P., & S. Kayalvizhi, R. R. (2010). Evolving a New Model (SDLC Model-2010) for Software Development Life Cycle (SDLC). *Internation Journal of Computer Science and Network Security*, 10(1) 112-119.

Rainer, R. K., Turban, E., & Potter, R. E. (2009). *Introduction to Information Systems (2nd ed.).* Hoboken, NJ: Wiley.

Rothi, J., & Yen, D. (1989). Systems Analysis and Design in End User Developed Applications. *Journal of Information Systems Education*.

Ruparelia, N. B. (2010). Software Development Lifecycle Models. *ACM SIGSOFT Software Engineering Notes*, 35(3) 8-13.

Sharma, B., Sharma, N., & Sharma, N. (2009). Software Process Imporvement: A Comparative Analysis of SPI Models. *International Conference on Emerging Trends in Engineering & Technology (ICETET)* (pp. 1019-1024). Washington DC, USA: IEEE Computer Society.

Shelly, G. B. (2012). *Systems Analysis and Design.* Boston, USA: Course Technology, Cengage Learning.

Sommerville, I. (2011). *Software Engineering.* Boston, Massacheusetts: Addison Welsley.

Stoica, M., Mircea, M., & Ghilic-Micu, B. (2013). Software Development: Agile vs. Traditional. *Informatica Economica*, 17(4) 64-76.