Software Engineering Methodologies: A Review of the Waterfall Model and Object-Oriented Approach

Adetokunbo A.A. Adenowo, Basirat A. Adenowo

ABSTRACT—This paper discusses two main software engineering methodologies to system development, the waterfall model and the object-oriented approach. A review of literature reveals that waterfall model uses linear approach and is only suitable for sequential or procedural design. In waterfall, errors can only be detected at the end of the whole process and it may be difficult going back to repeat the entire process because the processes are sequential. Also, software based on waterfall approach is difficult to maintain and upgrade due to lack of integration between software components. On the other hand, the Object Oriented approach enables software systems to be developed as integration of software objects that work together to make a holistic and functional system. The software objects are independent of each other, allowing easy upgrading and maintenance of software codes. The paper also highlighted the merits and demerits of each of the approaches. This work concludes with the appropriateness of each approach in relation to the complexity of the problem domain.

Index Terms—Object-oriented Approach, Software, Software Engineering, Software Engineering Methodologies, Software Objects, Traditional Approach, Waterfall Model

---

1 INTRODUCTION

Today, many computers or electronic systems run software to address scientific, social as well as economic problems. The importance of software—an abstract structure—in many facets of life call for an engineering approach towards its development, thus making it (i.e. software) the object of Software Engineering. Various definitions of software engineering have been proffered in the literature (see [6], [9]). Wang [17] defined software engineering as a discipline that studies the nature of software, approaches and methodologies for large-scale software development, and theories and laws behind software behavior and software engineering practices, aiming at high productivity, low cost, controllable quality and measurable development schedule. McDermid [8] defined software engineering as "*….the science and art of specifying, designing, implementing and evolving – with economy, timeliness and elegance – programs, documentation and operating procedures whereby computers can be made useful to man.*"

The foregoing definitions thus suggest the significance of adopting the most appropriate methodology and/or approach that yield the best results, thus necessitating the application of engineering principles. While the former definition perceives software engineering—in the context of nature—as an engineering discipline that adopts engineering approaches (methodologies, processes, measurements, tools, standards, organizational methods, management methods and quality assurance systems), with object under study being "large scale software" and aims the following attributes: productivity, quality, cost and time [18]. The latter definition portrays software engineering—in terms of nature—as science and art, adopted the means of life

* Adetokunbo A.A. Adenowo—the corresponding author—is currently the Ag. Head, Department of Electronic & Computer Engineering, Lagos State University, Lagos, Nigeria. Email: adetokyom@yahoo.com

* Basirat A. Adenowo—is currently a Senior Lecturer at the department of Computer Science, School of Science, Adeniran Ogunsanya College of Education, Oto-Ijanikin, Lagos, Nigeria. Email: b.adenowo@live.com

cycle methods (including specification, design, implementation and evolving), with the object of study being "program and documentation" and aims attributes of economy, reliability and efficiency [8].

Thus, software engineering could be said to involve both analysis and design of a software system that addresses a specific task or problem domain, and includes elaboration of concept(s) which will later be constructed or developed into appropriate software system(s). According to Bennett et al. [1], analysis describes the "what" of a software system, which means what happens in the current and what will be required in the new software system; this refers to requirement analysis or gathering. On the other hand, design describes the "how" of a software system; that is, how the system will be constructed. Thence, analysis and design make up the foundation upon which information system—an integrated set of components that includes the software element—is built; they (i.e. analysis and design) constitute major elements of software engineering. Satzinger et al. [15] also stated that System Development Life Cycle (SDLC), or alternatively, software development life cycle, is a very fundamental concept in information system development. SDLC is the process of creating or altering information systems, and the models and methodologies that could be used to develop these systems [14]. Software engineering thus makes available a number of methodological approaches that could be implemented during SDLC.

In recent time, the most popular methodological approaches for developing software for a computer-based information system are the popular traditional Waterfall Model [12] and the Object-Oriented approach [5]. The latter is sometimes considered a technique rather than a model. The waterfall model (or sometimes referred to as structured analysis and design model) follows sequential process and separates data in a system from the programs that act on the data. This traditional approach (i.e. Waterfall model) renders a rigid system development process [11]; hence, software systems—using this approach—are not easily upgradable or easily repaired. Hence, coupling between subsystems do occur—changing the processes if the data are to be changed. On the other hand, Object Oriented approach is based on the analysis and design of a collection of software objects, representing solution to a single problem or concept, that are integrated and work together in order to provide a holistic system functionality. The objects represent "*instances of programming constructs, normally classes, which are data abstractions and which contain procedural abstractions that operate on the objects*" [5, p.31] Thus, each object is an encapsulation of its states/attributes, behavior/operations and identity of the object.

It should be noted that both the Waterfall and the Object-oriented approaches depend solely on the understanding of system requirements in order to make meaningful elicitation during analysis and design. Hickey and Davis [3] affirms that knowledge of existing and proposed software system is important in performing requirements elicitation and only the selection of an appropriate elicitation technique (e.g. brainstorming, document analysis, focus group, interface analysis, observation, prototyping, survey, etc.) could result in a successful analysis. In this work, the above mentioned approaches—Waterfall model and Object-oriented approach—are discussed. It aims, not only to shed light on the two approaches, but also, to identify factors that will inform the use of either approach. The next section discusses the two methodological approaches, thereafter, the merits and demerits of each approach are highlighted.

2 THE METHODOLOGIES

This section provides a deeper understanding of the traditional approach based on the waterfall model and the object oriented approach using iterative and incremental models.

2.1 The Waterfall Model

The traditional approach to software development can be illustrated through the waterfall model which is time-tested and easy to understand. The waterfall model is a static model and it approaches systems development in a linear and sequential manner, completing one activity before the other. Fowler [2] affirms that waterfall style breaks up projects based on activities: requirement analysis, design, coding and testing. Pressman [13] identifies the activities as: communication (involving project initiation and requirements gathering), planning (estimating, scheduling and tracking), modeling (analysis and design), construction (coding and testing), and deployment (delivery, support and feedback). Pfleeger and Atlee [12] present the model as involving the following phases: requirement analysis, system design, program design, coding, unit and integration testing, system testing, acceptance testing and operation and maintenance. Summarily, the waterfall model could be said to involve the following phases: requirement analysis, design, implementation (i.e. coding), testing, and operation and maintenance (see fig. 1 below).

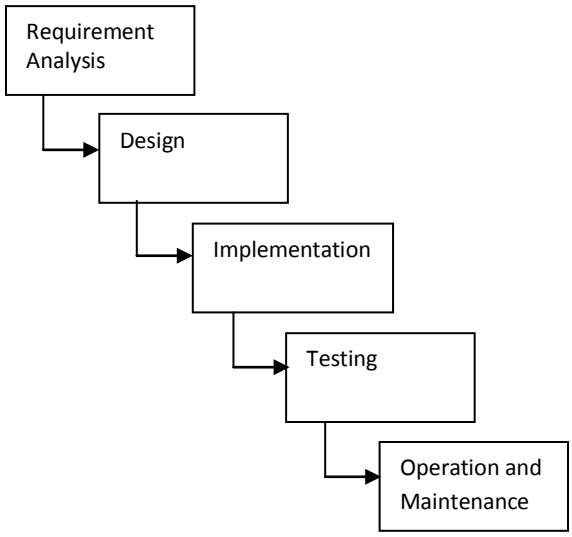

Fig. 1: The phases of a Waterfall Model (Adapted from Pfleeger and Atlee [12])

The waterfall model usually has distinct goals for each phase of development. Once a phase is completely developed, the development proceeds into the next phase and there is no opportunity to go back and revisit earlier stage as depicted in fig. 1 above. The model thus supports an approach that is structured and process-centered. Fowler [2] further stressed that there are usually some handoffs between phases and there are often backflows but they should be very much avoided. Any completed phase completes a particular set goal, which is quite different from the goal of the next phase. Also during design, if an error is detected in the completed phases, there is usually no opportunity to revisit the earlier phase. For instance, during design stage something may come up that requires you revisit analysis stage. It should be noted that during development process, an amendment may be necessary due to adjustment in requirement specification by the owner/user of the proposed system. Such amendment is impossible to achieve in waterfall development process; this depicts the weakness of the traditional approach.

Furthermore, in traditional waterfall model, development proceeds without any overlapping between stages. Although the model can accommodate iteration, it does so indirectly [13]. Once a phase is completed, there exists no room to revisit it over and over to detect any flaw. Thence, no improvements can be made since the phase cannot be revisited. This model is most useful in structured systems development where altering the software after coding is very much prohibited. Also, processes and data are usually separated in waterfall model, such that if the data are to be modified the code must be changed as well (known as software coupling). This makes software not reusable and system not easily upgraded because the entire processes will be modified in order to make any adjustment which can be cumbersome and expensive.

In recent times, there have been some improvements on the waterfall model which attempts to address the problems inherent in the traditional waterfall model. The improvements resulted in the Rapid Development

models that McConnell [7] calls "Modified Waterfalls". Unlike traditional waterfall model, the modified model allows phases of projects to overlap and still involves the phases in the traditional waterfall model: requirement analysis, design, implementation (or coding), testing and maintenance. Each phase in the modified model influences and depends on the next and the previous phase respectively (as depicted in fig. 2 below) and verification and validation has been added to each of the phases. The overlap of phases does provide flexibility in the software engineering process. Thus, it ensures that the defects in a software system are removed in the development stage, thereby reducing the overhead cost of making changes to a software project before implementation stage. Despite the overlapping of phases, a software project based on modified waterfall model is still prone to delay due to the dependency of a phase over the previous. Notwithstanding, this shortcoming could be eliminated by setting benchmark prior to commencement of the software project. As a result, many information systems and projects have adopted the modified waterfall model especially in the manufacturing and construction industries.

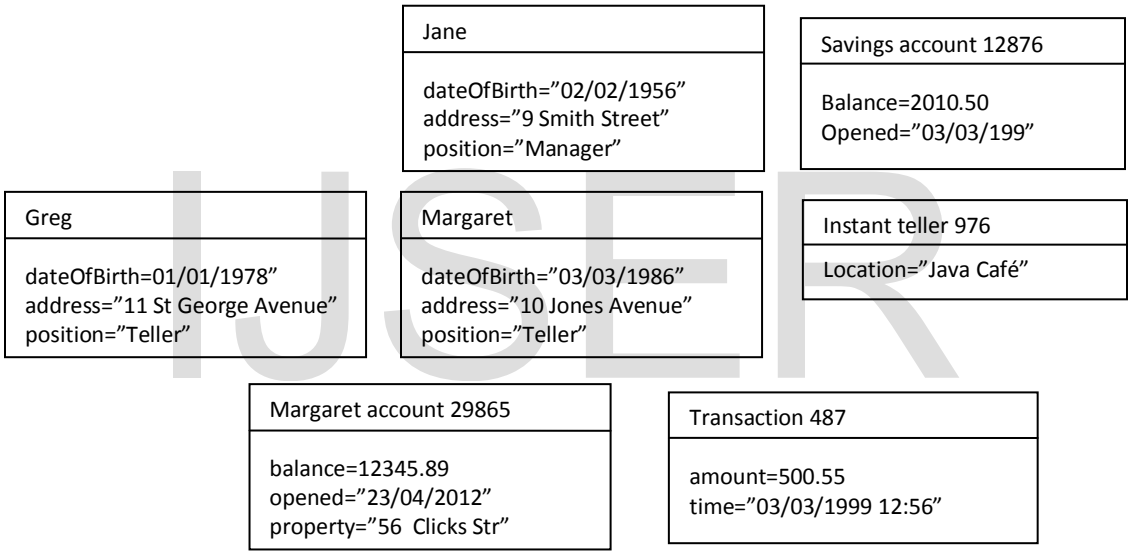

Fig. 2: Objects of a banking application (adapted from Lethbridge & Laganiere [5])

2.2 Object-Oriented Approach

Unlike the traditional system development model (such as the waterfall model) that regards processes and data as separate components, object-oriented approach models real-world processes using objects. That is, the solution of problems can be seen as a set of objects or computations performed in the context of objects [4], [5], [10], [11]16]. Data and the processes that act on the data are encapsulated within every object. Each object's data (attributes or states) are the properties that relate to the object. The object's operations are processes performed to modify the data in order to meet specific needs. An object can represent actual people, things, transactions, and so on. A software object is an instance of a class, and a class is a user-defined data type. A set of objects describe a class while each object consist of a set of properties. For example, in a result-computation system, the name of a class could be Student and names of the students (e.g. "Jones", "Chloe") could be two instances (two objects) of the Student class. In an organisation, department could be a class and the title of the departments (e.g. "admin", "works") could be object instances of the class. The fig. 2 above represents several objects and their properties in a banking application.

A class has both internal and external definitions. The external definition of a class is the class interface through which objects of other classes and programmers of those objects are able to know services rendered by the objects of that class and the signature to request the services. Therefore, access to the data within an object by other objects is available only via the objects' interface. The internal definition of a class refers to what the objects of that class know and what they can do. Only objects of a class know the internal definitions of the class. Internal definition of a class ensures good code modularity, meaning that less programming is required when adding new functions to the complex systems.

In Object-oriented development, information system is constructed so that the implementation of each part is quite independent of the implementation of the other part (decoupling of software), due to possibility of modularization. Each software object is coded and implemented and then integrated to the Information System. This continues until the entire Information System is completed. So, there is decoupling of software because each software object can be modified and recoded and its data adjusted without disrupting the entire system. Due to modularisation, each process is located with the data it uses and this gives ample opportunity for reuse of software components. The entire system is constructed as integration of software objects with each object consisting of the processes and set of data that they work on.

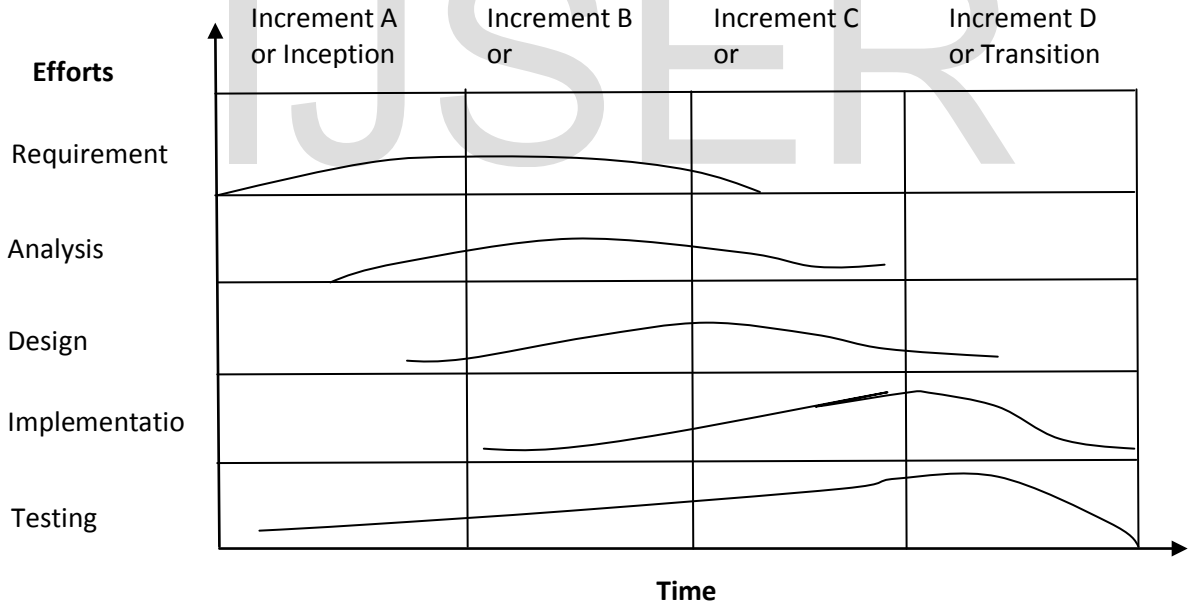

Fig. 3: Object-oriented method using increment and iteration approaches (adopted from Rob, M., 2004)

Object Oriented approach solved the problem of structured SDLC (e.g. waterfall model) by using iterative and incremental models, so earlier stages can be revisited and earlier products revised by repeating the processes until good and quality software is produced. The Object-oriented SDLC is viewed as consisting of increments or phases: inception, elaboration, construction and transition, and each increment or phase implements the stages in traditional structured models: requirement analysis, design, implementation and testing [14] (see fig. 3 above). Satzinger et al. [15] describes the iterative and incremental approach as a

spiral model that cycles the development activities over and over again, leading to improvement as project progresses. The actual concept here is to develop a system through repeated cycles (iterations) and in smaller portions at a time (incremental), allowing software developers to take advantage and learn from both the development and use of the system. At every iteration, design modifications are made and new functional capabilities are added until the entire process is completed.

3 MERITS AND DEMERITS

This section compare and contrast the two approaches discussed above, highlighting their merits and demerits.

3.1. Merits & Demerits of Traditional Waterfall Approach

The waterfall approach still remains the most popular model used in the field of software development. Being a linear and structured model, it is very simple to implement, and less expensive. Bennett et al. [1] identified some advantages of waterfall: that it is good for effective control and management of resources such as money, staff and time. For instance, merging analysis and design may be cumbersome if staff skills and experience require separating analysis stage from design stage. It is mostly used in industries for development of software that is expected to flow steadily downwards like a waterfall where highly structured programs are needed and in which changes after coding are prohibitively costly, if not impossible. Documentation is also produced at every stage of the software development, which enhances understanding the product designing procedure.

The most obvious disadvantages of the traditional waterfall model are the inability to evaluate the outcome of one stage before moving on to the next (intermittent evaluation) and the inability to go back to any step to make changes in the system. Sometimes, the client is not very clear of what he exactly wants from the software, so mentioning any changes in between may cause a lot of confusion. The entire process is sequential and there is no opportunity to revisit the previous phase. Thus, he is hardly in a position to inform the developers, if what has been designed is exactly what he had asked for. Lack of integration between software components and separation of processes from data are other disadvantages of the waterfall which makes it unsuitable for Object-Oriented programming. There is coupling of software components causing software to be reworked if data are to be changed, this makes software not reusable.

3.2 Merits & Demerits of Object Oriented Approach

Since the Object Oriented method makes use of iterative and incremental steps, it gives opportunity to manage changes as they occur to user requirements. So, it is more prone to user satisfaction. Due to several iterations of an increment, potential risks are quickly and easily identified, and new codes reworked while existing ones are deleted. Another advantage of the Object Oriented method is that it gives room for iteration retrospect and opportunity for the team to learn in the process, as such design modifications can be made and new functional capabilities added. Successful iterations imply completion of an increment which means production of a subsystem needed for specific functionality. This provides feedback to the development team whether to move to subsequent increments.

Furthermore, the fact that software objects are encapsulated as a result of modularisation, make system easy to maintain, easy to upgrade, and more reliable. There is opportunity to reuse software components since they are very much decoupled with low degree of dependency of program modules on each other. The major disadvantage of Object Oriented approach is, not knowing when exactly to stop iterations. The

development team may be tempted to remain in several loops of iterations still wanting to come up with a perfect functioning system. Another disadvantage of using Object Oriented method is that it can be very expensive. It is also difficult to come up with an object-oriented system because it is very time-consuming and cumbersome.

4 CONCLUSIONS

From the above discussions, Object-oriented method is a very flexible approach tolerating changes and improvements throughout system development due to the style of continuous chain and cyclical model. The traditional waterfall is more rigid because of its linear approach, and there may be little or lesser user satisfaction since there is no opportunity to make changes to the system. As such, the quality of deliverables of Object-oriented development is very high and robust compared to traditional waterfall-based systems which are mostly error prone. However coming up with an Object-oriented system can be difficult and also quite expensive. Waterfall on the other hand is simpler to implement and less expensive, that being the reason it is more widely used, especially its modified version. None the less, once an Object-oriented system is developed it can be reworked easily to improve the existing system, and can be reused severally for other applications with little adjustments.

Therefore, it can be concluded that the two approaches are still functional in system development, but Object-oriented method is more efficient and effective, facilitating better user satisfaction of information systems than the waterfall. The object-oriented approach thus tends to have an edge over traditional waterfall model in that it is readily applicable to real world problems, reducing complex problems to a collection of integrated objects, grouped into classes with associated relationships. Thus, when the focus is to model complex problems that will require revisit of previous phase(s) , to attend to changing requirements or address issues that emerge during the development process, then object-oriented appears more appropriate than the traditional waterfall model. However, when the problem domain and requirements are very clear and straightforward, the traditional waterfall model could be easily adopted due to its simplicity and sequential process.

REFERENCES

[1] Bennett, S., McRobb, S. & Farmer, R. (2002). Object- Oriented Systems Analysis and Design Using UML Berkshire: McGraw-Hill Education.

[2] Fowler, M. (2004), UML Distilled a Brief Guide to the Standard Object Modelling Language, Boston: Pearson Education, Inc.

[3] Hickey, A. M & Davis, A.M. (2003). Requirements Elicitation and Elicitation Technique Selection: Model for Two Knowledge - Intensive Software Development Processes. In: Systems Sciences, Proceedings of the 36th Annual Hawali International conference, 6-9, Jan., 2003.

[4] Larman, C. (2005). Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development. New Jersey: Pearson Education, Inc.

[5] Lethbridge, T.C. & Laganiere, R. (2005). Object-oriented Software Engineering: Practical Software Development using UML and Java, 2nd edition. UK: McGraw-Hill.

[6] Mathew, S. (2007). Software Engineering. New Delhi: S. Chand & Company Ltd.

[7] McConnell, S.M. (1996). Rapid Development: Taming Wild Software Schedules. Microsoft Press.

[8] McDermid, J.A. ed. (1991). Software Engineer's Reference Book. UK: Butterworth-Heinemann Ltd., Oxford.

[9] Mnkandla, E. (2009). About Software engineering Frameworks and Methodologies. Proceeding of IEEE AFRICON 2009, 23-25 Sep., 2009, Nairobi, Kenya.

[10] Munassar, N.M.A & Govardhan, A. (2010). A Comparison of Five Models of Software Engineering. International Journal of Computer Science, 7(5).

[11] Munassar, N.M.A & Govardhan, A. (2011). Comparison between Traditional Approach and Object-oriented Approach in Software Engineering Development. International Journal of Advanced Computer Science and Applications, 2(6).

[12] Pfleeeger, S.L. & Atlee, J.M. (2006). Software Engineering: Theory and Practice, 3rd Edition. US: Prentice Hall.

[13] Pressman, R.S. (2005). Software Engineering: A Practitioners Approach, 6th Edition. Singapore: McGraw-Hill.

[14] Rob, M.A. (2004). Issues of Structured Vs. Object-oriented Methodology of Systems Analysis and Design. Issues in Information Systems, 5 (1).

[15] Satzinger, J.W., Jackson, R.B. & Stephen, D.B. (2008). Systems Analysis and Design in a Changing World, Lengage Learning EMEA 3rd Edition .

[16] Schach, S.R. (2004). Introduction to Object-oriented Analysis and Design with UML and the Unified Process. New York: McGraw-Hill.

[17] Wang. Y. (2008). Software Engineering Foundations: A Software Science Perspective. New York: Taylor & Francis Group.

[18] Wang, Y. & King, G. (2000a). Software Engineering Processes: Principles and Applications, CRC Book Series in Software engineering, Vol. 1, CRC Press, USA.