A SURVEY OF SOFTWARE ENGINEERING MODELS, COMPARISONS AND SCENARIO OF PROJECTS

Supported by tetfund

A. O. Adesina*, T. J. Odule, Q. A. Alatishe and O. A. Morafa
Department of Mathematical Sciences (Computer Science Unit), Olabisi Onabanjo University, Ago Iwoye, Nigeria

Received: July 13, 2020 **Accepted:** September 02, 2020

Abstract: The choice of appropriate model for the development of software product is a paramount factor in the field of Software Engineering (SE). System development and system testing processes are carried as a part of SE phase. The software process model is a format all activities carried out during the production of the software which includes planning, organizing and running the project. A process activity model depicts events as well as associated sequences though may hide functions of participants concerned. Accordingly, this paper reviews the present opportunities in the process model with the view to recommend the appropriate model for a peculiar software development project by making a survey and summarizing the existing software development life cycle (SDLC) models.

Keywords: Software development, system life cycle, software development models, SDLC

Introduction

A systematic approach to the crafting of computer programs represents software engineering, often a systematic collection of past experience to solving a software developmental process (Jalote, 2012; Mall, 2018). Software development model is also referred to as Software Development Life Cycle (SDLC) concept or Software Developmental Process Representation. It describes an abstract structure representing the totality of events during an application or a system program developmental project spanning between preparation and on-the-job support. SDLC has numerous representations, with each having an assortment of events and responsibilities (Akbar *et al*., 2017). Software development remains a tedious process that demands correct documentation of system interactions and needs, execution methods as well as program installation. More so, the activity further goes to packaging the suite of programs and appropriate support must be given as at when required.

The product representations constitute numerous procedures or techniques chosen for the design and implantation of the job consequent upon its target and objectives (Joslin & Müller, 2016). Numerous software developmental process representations exist proposed to accomplish distinct requisite intents. These representations determine the different phases in product development as well as the request for handling them. Fruitful programming teams need to find some kind of harmony between rapidly conveying working programming frameworks, fulfilling their partners, tending to potential threats, while refining the applied methodology. This necessitates an operational responsive system, i.e. models which can absorb changes in implementation paradigm (Jacobson *et al*., 2012; Jacobson & Stimson, 2017).

There are a wide range of application development models yet each should incorporate four exercises deemed essential to application designing:

1) Software specification – lists usefulness of the application together with all activity-related restrictions well defined.

2) Software design and implementation – application that is consistent with requirements should be delivered.

3) Software validation – proper certification of the application to make it consistent with client expectation, and

4) Software evolution – application developed such that it adapts to varying client requirements.

These application succession states may contain complex and sub-complex events like certifying client specifications, formulation of blueprint, individual component validation, etc. Auxiliary method events like proper record-keeping as well as tracking and controlling changes in the application, all exist.

Importance of software process models

Every software development organisation adheres to some models which assist on what to do, how to do and when to do it to avoid conflict and possible software development failure. Some of the important software process models include:

i. Promotion of better communication among stakeholders.
ii. Production of better quality invention and documentation standards.
iii. Assurance of user requirements should be fulfilled.
iv. Support for leader via offering of enhanced regulation of job implementation, including a decline in total debt budget.
v. Encourage understanding of the system through standardization of process and documentation.

Stages of SDLC

There are six identifiable stages involved in the SDLC. They are:

1. Requirement gathering and analysis: This stage is for gathering trade requirements. It constitutes the main interest area to both the project controllers as well as associates. Specifications collated are scrutinized to ensure legitimacy with the prospect of integrating such requirements with the structure being modelled for developmental purpose.
2. Design: This is where the software designer introduces the approach for designing and testing every stage of the software developmental process with emphasis on the part of the system designed to be tested and how to be tested.
3. Implementation/coding: In this stage, the application is shared into functionally independent parts where in real programming start. This phase constitutes most important and tasking phase for the programmer.
4. Software testing: This part checks the application structure is test alongside practical necessities and requirements. This is often achieved by feeding the application with data while checking it for corresponding result. This checking methodology guarantees adequate representation of specifications by the software. Functional testing is a value guarantee method, a sort of black-box test in which the functionality of an application is tested without looking at the internal code structure, implementation details and knowledge of internal paths of the software (Howden, 1987).
5. Software distribution: Once the application is decisively debugged, it is conveyed/distributed for client utilisation (Carzaniga *et al*., 1998). It includes installation, configuration, testing and making changes to streamline the presentation of the software (Dolstra, 2006).

6. System maintenance: it's the modification of a system to correct faults to boost performance, or customise it to an adjusted setting or modified needs. Once the application is being utilised by the client, the day-to-day challenges manifest, which will require regular attention (Yeddanapudi *et al.*, 2008).

Software process models adopted by organisations

Several software development paradigms exist but some known ones include:

Build and fix model (ad hoc model) The product is created with no requirement and structure. An ad-hoc approach which is not well defined is utilized by the programmer (Davis *et al*, 1988). A preliminary artefact is constructed, and then adjusted till it fulfills its purpose. The model is not organized and no prior planning is established before the software development.

Waterfall model This is the oldest SDLC paradigm in use for application development. It is a sequential development concept (Weisert, 2003). Fig. 1 is a pictorial representation of the application paradigm. Commencement of a stage in this paradigm implies the completion of, and no association with, the preceding stage.

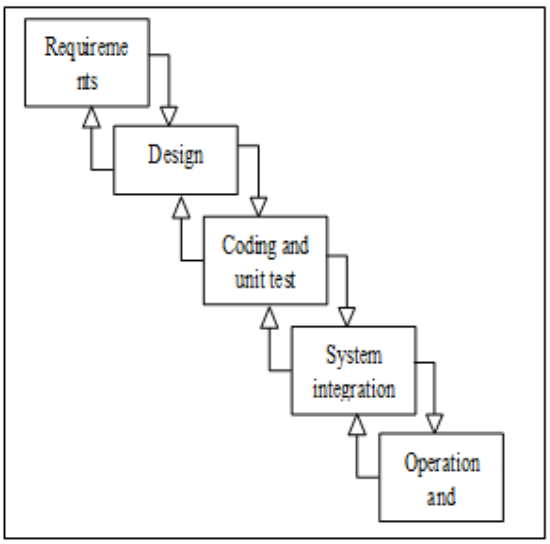

Fig. 1: Waterfall model

Prototyping model An early sample or model built to test a concept or process is called a prototype (Blackwell *et al.*, 2015). Prototyping allows customers opportunity to evaluate the application developer's first-hand understanding of the application's requirements with a chance to try out some of its features prior to its full implementation. It is a miniature representative software having limited functionality. The prototype is not reflective of the actual logic used by the final software and is an extra drive factored into effort valuation. Fig. 2 shows the prototype model work flow.

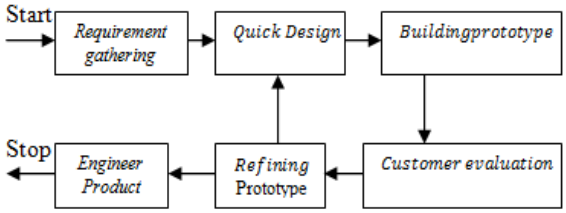

Fig. 2: Prototyping model

There exists a loop in the sequence of operation of prototyping model. The refined prototype can finally be adopted by the engineer it enters the quick design phase.

Incremental model This model is best for when a customer want some changes in the product (Craig *et al*., 2003). Fig. 3 shows the incremental representation which starts out on a modest execution of a lesser part of the application specification, repetitively improving developing version till the total framework is actualized as well as fit distribution and installation. This representation never starts out on a detailed description of design requirement. Rather, advancement begins via introduction and executing parts that make up the application, that are later looked into for recognition of additional requests.

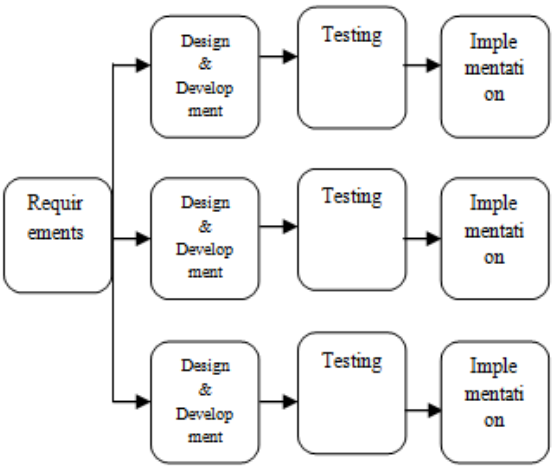

Fig. 3: Iterative/incremental model

Spiral model This representation combines likelihood of repetitive execution advancement with systematic, well-ordered portions featured in the sequential representation, allowing gradual product release or gradual refinement via the cycles within the enclosing helical curve. Each traversal performed by the spiral usually produces a deliverable outcome of the project. Fig. 4 shows the Spiral model.

Generally, this representation endeavours uniting essential elements in selected notable paradigms (in particular, waterfall, incremental, as well as prototyping) with the purpose of featuring the best characteristics inherent in every one, since particular applications may be pretty much versatile to particular paradigms (Aggarwal and Yogesh, 2007).

Fig. 4: Spiral model

Agile model Fig. 5 describes the Agile representation as amalgam of iterative and incremental procedures whose pivots centre around task flexibility with consumer fulfillment

through quick provision of functional application. This method breaks the application to lesser gradual constructions (Cohen, 2003). This paradigm is characterised by a group that is adjustable and imbued with ability able to cope with dynamic requests (Marti, 1999).

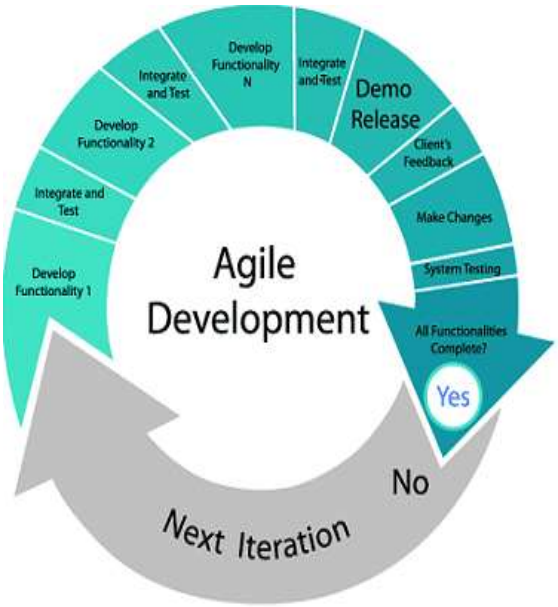

Fig. 5: Agile model

Rapid application development (RAD) RAD paradigm fuses prototyping and iterative advancements without specific designing involved. During the developmental stage, the knowledge gained can be used to produce a preferred solution (Brooks *et al*., 1986). An approach to creating the application entails the planning necessary in structuring the application. Fig. 6 shows the RAD paradigm flow.

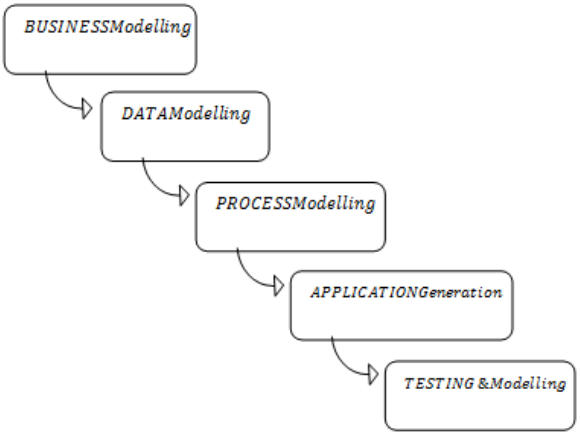

Fig. 6: RAD model

Comparison of the SDLC models

Table 1 depicts the comparison of the SDLC models emphasising their strengths, weaknesses, types of projects best used for and the project examples.

Table 1: Models strengths, weakness, project types and example of the project

SDLC paradigm	Merits	Demerits	Scenario of projects
Sequential Paradigm	1. Expanded documentation is done at each period of the product's advancement cycle 2. There is simplicity and ease in the usage of the model. 3. It is in line with many innovative customs 4. It reduces preparation burdens.	1. Result delivery is late in the developmental cycle. 2. The model doesn't bolster iteration and it is firm. 3. Errors/changes in the completed work are difficult to debug in this model 4. The user feedbacks are not taken during development.	1. Relatively fixed specifications 2. Simple and moderate application 3. Environment is stable 4. Resources are available and trained like Design for small company websites, Governmental projects, Healthcare projects etc.
Iterative / Incremental Model	1. It saves time as software is created rapidly. 2. This model is manageable and inexpensive in altering specification and depth of coverage. 3. User feedbacks are supported. 4. Amenable to alterations all through the phases of application creation 5. Accommodates client's reaction to every partial construction	1. This model has stages that are inflexible and do not overlap. 2. Problem related to system architecture in future iteration may occur. 3. The paradigm demands effective preparation as well as designing.	1. Application specifications unambiguously comprehended. 2. Need for short turnaround time for application delivery. 3. If application developers are amateurs or unskilled. 4. If application contains great premium characteristics as well as targets. For projects like Enterprise applications such as micro services or web services, Electronic commerce website or Portal.
Agile Model	1. There is great manageability in job handling. 2. There is great client fulfilment concerning the creation procedure 3. Continuous communication amid the participants. 4. Constant assessment of specifications with requirements, devotion to specifics.	1. Task completion sequences are difficult to harmonise and manage. 2. Challenging preparation during initial phases. 3. Professional teams are very important for decision making. 4. Absence of extensive preparation.	1. Fair sized activities in custom programming advancement where business requirements cannot be translated into software requirements. 2. Large projects divided into small functional parts. For Large and complicated or unclear projects like building a social network.

SDLC paradigm	Merits	Demerits	Scenario of projects
Build and Fix Model	1. This model requires little coding experience from the developer 2. The model is good for small projects 3. The model requires little or no planning	1. There is no valid quality and progress control. 2. The model takes time and high cost is incurred. 3. The software design is informal. 4. The maintenance is complicated as there is no proper planning	1. Mini projects and programming exercises such as proof of concept, demos and software prototypes.
Prototyping	1. In this model, clients keenly participate in application creation. 2. Omitted tasks and functions are easily highlighted, thus aiding a decrease in the likelihood of disappointment. 3. Helps team member to communicate effectively 4. Client fulfillment is guaranteed since the application may be experienced from the onset. 5. The model encourages innovation and flexible designing.	1. The paradigm takes a gradual as well as protracted procedure 2. Prototypes developed are mostly thrown away. 3. The model is prone to errors 4. Defective operational guide due to variations in user specifications.	1. When there are uncertainties in requirements. 2. For projects like Website development and web application development such as social media.
Evolutionary Model	1. It encourages you to spare time and effort 2. The client can explore the framework to improve the requirements the requirements 3. Customer's feedback is supported	1. Embodies greater premiums, hence, effective supervision and control very essential. 2. The paradigm is employed as alibi for security control to evade recording the requirements or style, even though they're well understood.	1. The paradigm is useful where an application utilises unfamiliar novel innovation. 2. Utilised in a multifaceted application whose working order needs immediate verification 3. It is helpful when the requirement is not stable or not understood clearly at the initial stage like Resource management projects of all kinds.
Spiral Model	1. Further modifications may be effected in a further phase. 2. Price valuation is straight forward because model construction completes in piecemeal 3. It reduces likelihood of failure from the onset thus avoiding possible breakdown. 4. Client reaction is incorporated into the design procedure.	1. Usually applied to lesser applications in view of development overhead expenses. 2. Probability of failed completion deadline or exorbitant development cost. Failure in the absence of listening and effective supervision and control.	1. When releases are required to be frequent 2. If assessment of likelihood of failure as well as budgets are essential 3. For moderate to greater premium applications 4. For specifications that are ambiguous and intricate 5. If unscheduled alterations must be accommodated whenever. 6. If extended assurance of application is unrealistic because of vagaries in monetary exigencies like Research and development projects, large and complicated projects (unclear projects or websites).
Extreme Model	1. The model saves cost and time. 2. It is very simple to build 3. Encourages transparent, explainable as well as traceable procedures 4. Constant feedback is also supported and this yield to better customer satisfaction.	1. The model concentrates on programming in contrast to blueprint. 2. It falls short of programming standard assessment 3. It is not a good practice when programmers are separated geographically.	1. Prerequisites for the entire structure usually unidentified at the outset 2 Producing simple application. 3. Designers and programmers are closer to one another. For building responsive websites like social network, e-commerce websites.
RAD (Rapid Application Development) model	1. Amenable as well as adjustable to accommodate alterations. 2. The model becomes handy for lessening general likelihood of application failure 3. Encourages rapid early appraisals. 4. Enhances client feedback. 5. With less individuals, productivity are often increased in short time	1. Requires effective group as well as personal feats in recognising corporate prerequisites. 2. Especially suitable for constructing adaptable applications. 3. Necessitates the use of seasoned professional programmers. 4. Greatly relied on modelling abilities. 5. Shorter completion deadline may trigger crisis	1. When anapplication must be created relatively quickly. 2. If application prerequisites are well-understood. 3. If client participation is assured during the entire stages of application development. 4. When technical risk is less 5. When a budget is high enough. For projects like Employee management system, purchase order projects, and so on.

Table 2: Features and recommendation model

Features	Recommended model
i. Reliability	Spiral, Iterative/Incremental, Agile
ii. Stable funds	Waterfall, Prototyping, Iterative/Incremental, Spiral, RAD, Evolutionary, Extreme, build and fix
iii. Tight project schedule	Prototyping, Spiral, Iterative/Incremental, Evolutionary, Extreme, RAD.
iv. Scarcity of resources	Prototyping, Spiral, Iterative/Incremental, Evolutionary, Extreme, Agile.
v. Changes in requirements	Prototyping, Spiral, Iterative/Incremental, Agile.
vi. Limited user involvement	Waterfall, Spiral
vii. Constant feedbacks from users	Prototyping, Iterative/Incremental, RAD, Evolutionary, Extreme, Agile, Build and fix.
viii. Little experience	Build and fix.

Basic criteria for selecting a software development model

A proper comprehension of an application's prerequisites with regard to scope, intricacy, accessible monetary resources risks involved, etc. constitutes a major highlight in choosing a paradigm. Table 2 depicts the features and recommendations for selecting particular model to be used for software development.

Summary and Conclusion

Selecting a model for software development project seems to be a challenging task due to the various pros and cons. This paper focused on the major and widely used models; their strengths, weaknesses, application areas, so that developers would be able to select a better model best fit for project of choice.

This paper focused on some basic criteria. Other criteria in relation to various application domains (such as medical sector, banking sector, educational sector) can also be considered as various domain has its own requirements and functionalities.

Conflict of Interest

Authors have declared that there is no conflict of interest reported in this work.

References

Akbar MA, Sang J, Khan AA, Shafiq M, Hussain S, Hu H & Xiang H 2017. Improving the quality of software development process by introducing a new methodology-AZ-model. *IEEE Access*, 6: 4811-4823.

Blackwell AH & Manar E 2015. Prototype. *UXL Encyclopedia of Science* (3rd ed.).

Brooks Fred 1986. Kugler HJ (ed.). *No Silver Bullet Essence and Accidents of Software Engineering* (PDF). Information Processing 86. Elsevier Science Publishers B.V (North-Holland). ISBN 0-444-70077-3.

Carzaniga A, Fuggetta A, Hall RS, Heimbigner D, Van Der Hoek A & Wolf AL 1998. A characterization framework for software deployment technologies: Colorado State Univ Fort Collins Dept of Computer Science.

Craig Layman & Victor Basili 2003. Iterative and Incremental Development: A Brief History. *IEEE Computer*.

Davis M, Bersoff H & Comer ER 1988. A strategy for comparing alternative software development life cycle models. *J. IEEE Transac. on Software Engr*., 14: 10.

Dolstra E 2006. *The Purely Functional Software Deployment Model*. Utrecht University.

Howden WE 1987. *Functional Program Testing and Analysis* (Vol. 2): McGraw-Hill New York, NY.

Jacobson I, Ng PW, McMahon PE, Spence I & Lidman S 2012. The essence of software engineering: the SEMAT kernel. *Communications of the ACM*, 55(12): 42-49.

Jacobson I & Stimson R 2017. Escaping method prison–On the road to real software engineering. Springer, Cham, *The Essence of Software Engineering*, pp. 37-58.

Jalote P 2012. *An Integrated Approach to Software Engineering*. Springer Science & Business Media.

Joslin R & Müller R 2016. The impact of project methodologies on project success in different project environments. *Int. J. Managing Projects in Bus*.

Mall R 2018. *Fundamentals of Software Engineering.* PHI Learning Pvt. Ltd.

Yeddanapudi SRK, Li Y, McCalley JD, Chowdhury AA & Jewell WT 2008. Risk-based allocation of distribution system maintenance resources. *IEEE Transac. on Power Syst.*, 23(2): 287-295.