An Excursion to Software Development Life Cycle Models: an Old to Ever-growing Models

Unnati S. Shah
Department of Computer Engineering
C. K. Pithawalla College of Engineering and Technology
Near Malvan Mandir, Dumas Road, Surat-395007
unnati.shah25@gmail.com

DOI: 10.1145/2853073.2853080

ABSTRACT http://doi.acm.org/10.1145/2853073.2853080

Software Engineering provides a standard way to develop and maintain a complex software. Industry uses software development Life Cycles (SDLC) to develop a software. SDLC plays an important role as it helps to define the software requirements, model the software component, reduce development and maintenance cost and finally provides manageable software. There exist numerous SDLC models viz. Waterfall, Incremental, Rapid, Agile, Hybrid etc. After a comprehensive study and analysis of existing SDLC models, I observe all models are complementary, not competitive. I divide all models into three broad categories viz. Traditional models, Agile models and Hybrid models. The main objective of the paper is to give a quick review of SDLC models and an effective answer to the most confusing question arise in software engineering practice "how to select an efficient SDLC model for practice?" Many factors viz. nature of requirements, the size of software development team, project size, customer interaction etc. have an effect on selection criteria. This paper presents a brief insight into each model and its comparative analysis. The analysis helps to understand the basic characteristics of each model and its applicability. Furthermore, the analysis helps software manager to select the appropriate model for practice.

Categories and Subject Descriptors
D.2.9 [**Software Engineering]**: Management – Life cycle.

General Terms
Management, Documentation, Performance, Design, Theory.

Keywords
Software Engineering, Software Development Life Cycle, SDLC models, Traditional SDLC models, Agile models, Hybrid models

SOFTWARE DEVELOPMENT LIFE CYCLE

Software Development Life Cycle includes various phase viz. requirements gathering, design, implementation, testing, deployment and maintenance [1, 2, 3, 4, 5]. Usually, software developers avoid using systematic way to build the software. The avoidance may lead to implementation and management issues that ultimately increase the cost. Various SDLC methods are available in the literature [1, 2, 3, 4, 5, 6] that can be broadly divided into three categories viz. Traditional models, Agile models and Hybrid models.

Traditional SDLC Models

A traditional waterfall model [5] and its variants includes Verification and validation model (V-shape Model) [7], Incremental model [8], Rapid development model (RAD) [9], Iterative model [10], Evolutionary model [11], Prototype model [12] and Spiral model [13]. A linear waterfall model ensures design flows before development. It is applicable if requirements are well-known, unambiguous and stable. However, the model has downsides viz. unrealistically expects accurate requirements at early stage - not suitable for real projects, expensive to make changes to documents due to rigid linear nature, high cost and administrative overhead. A modified waterfall model-Verification and Validation model (V-shape) - provides proactive testing at each phase. The model finds error as early as possible in the life cycle thus save both the time and cost. However, due to rigid linear cycle same as the waterfall model it suffers from problems viz. no early prototypes of the software, midway changes affect the time and budget, high risk meeting customer's expectation. Small to medium sized projects where stable requirements and technical resources are available one can use V-shape model.

The major limitation of the waterfall and V-shape models is not to produce an early prototype that helps to understand complex system requirements. To overcome the limitation an incremental model is developed. It produces early working modules where system functionalities are sliced into increments. The model starts with a partial implementation of a total system and then slowly adding increased functionality or performance with the assumption that all requirements understood properly and simply choose to implement in subset to increase system capability. It also produces an operational system more quickly, and thus reduces the possibility that the user needs will change during the development process. It is flexible and cost-effective because it easily tracks defects and use customer feedback at each stage. However, to divide whole system functionality into units/deliverable modules require unambiguous, complete system requirements (heavy documentation), good planning, and design at an early stage. Furthermore, the cost is higher compared to previous models viz. waterfall and V-shape models due to integration overhead. This model is applicable to an environment where requirements are well-known, unambiguous and complete, though evolving. The incremental model suitable for an environment where requirements can change with time, early delivery of product, new technology is being used and high-risk are associated. Another incremental model is a rapid model that focuses on the quick development of a system where system functionality is divided into modules. It requires high-skilled engineers. However, it fails if time constraints are not meeting. Furthermore, modularization is difficult for all system. Another variation is an iterative model that builds a supporting framework of available requirements, iterates the design step by step and tracks the defects at early stages. The model is applicable to the large system. On the other hand, the evolutionary model divides the system into small working deliverable components. However, to maintain progress and produce documentation that track modification and relationship of released version is bit difficult. To deal with large and mission-critical software projects spiral model is used that manage high

amount of risk. It uses a series of prototypes in stages and the development ends when the prototypes develop into a functional system. On the other hand to understand complex requirements prototype model is used that quickly develop the partial model as per the developers understanding and collected feedback from the user.

All discussed variants of traditional waterfall model encourage customer feedback and risk analysis. However, downsides of the models are less documentation, complex designing/modularization and high cost compare to waterfall and V-shape models. We can use these models where major requirements are clear, unambiguous and complete at an early stage though some details can evolve with time. However the basic difference between the prototype model and evolutionary model is the prototype model implements only those aspects of the system that are poorly understood, contrary the evolutionary prototype are more likely to start with those system aspects that are best understood [1, 18]. Table1. shows a comparative analysis of traditional SDLC models with comparison criteria viz. Workflow (linear or non-linear), Development approach (predictive / adaptive), Learning process (at which phase), Availability of RS (Requirements Specification at initial level), Nature of RS (fix/complete, evolving), Project size suitable for the model, Focus of the model, Level of documentation (high, moderate, less), Overlapping of phases, Evaluation at each phase, Early prototype (yes, no), Testing level, Beginning and End cost (moderate, less, high), Simplicity, Flexibility (rigid or moderate) and Management required, how much risk involve, how much customer involvement, expertise required, guarantee of success (uncertain, less, high, moderate), process time, how much incorporate changes of requirements, deliverability of component at which level, how many levels of customer satisfaction, how much development team size required, product delivery time (late, middle, quick).

Table 1. Comparative analysis of Traditional SDLC models

Features	**Waterfall**	**V-shape**	**Incremental**	**RAD**	**Iterative**	**Evolutionary**	**Spiral**
Work flow	Linear Sequential	Sequential	Multi-Sequential	Incremental	Iterative	Iterative	Incremental +iteration
Development Approach	Predictive	Predictive	Predictive/ Adaptive	Predictive/ Adaptive	Predictive/ Adaptive	Predictive/ Adaptive	Predictive/ Adaptive
Learning process	Beginning	Each phase	Each iteration	Each module	Each iteration	Each iteration	Each iteration
R:initial stage	Complete	Complete	Complete	Periodically release	Incomplete/ some part	Complete	Complete
RS: nature	Stable	Stable	Evolving	Evolving	Evolving	Evolving	Complex /Evolving
Project size/type	Small	Small	Large	Support Modularizatio n ion	Large	Small	Large/ critical
Focus	Complete development	Parallel testing	Modularization	Quick development	Creating a working prototype	Meet the project dead line/major feature in time	Risk analysis at each iteration
Documentation	High	High	Moderate	High	Moderate	Less	Less
Overlapping phases	No	No	No	No	No	No	Yes
Evaluation: Each phase	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Early Prototype	No	No	Yes	Yes	Yes	Yes	Yes
Testing	End	Each phase	Each iteration	Each module	Each phase	Each iteration	Each iteration
Beginning cost	Moderate	Moderate	Less	High	Less	High	High
End cost	Moderate	Moderate	Moderate	Moderate	Moderate	Moderate	Moderate
Simplicity	Yes	Yes	Moderate	No	Moderate	Less	Less
Flexibility	Rigid	Rigid	Moderate	Moderate	Moderate	High	High
Manageable	Yes	Yes	Yes	Moderate	Yes	Moderate	Moderate
Risk	High	Moderate	Less	Moderate	Less	Less	Less

involvement							
Customer interaction	Very less/ beginning	Very less/ beginning	Moderate	High	High	High	High
Maintainer	High	Moderate	High	High	High	Less	Less
Expertise re.	High	High	Moderate	High	Moderate	High	Very High
Guarantee of success	Uncertain	Moderate	High	High	High	High	High
Process time	High	High	Moderate	Less	Moderate	Less	Less
Change incorporated	Difficult	Difficult	Easy	Easy	Easy	Easy	Easy
Deliverable component	End	End	Each iteration	Quick	Uncertain at each phase/ further improved	Quick	Moderate
Integrity	Problematic	Problematic	Manageable	Manageable	Manageable	Manageable	Manageable
Client satisfaction	Uncertain	Uncertain	Achieved	Achieved	Achieved	Achieved	Achieved
Size of development team	Large	Large	Small	Large	Large	Large	Large
Product delivery	Late /behind schedule	Late / behind schedule	Middle	Quick	Middle	Quick	Quick

Agile SDLC Approaches

Software engineering research being targeted as real industrial problems. Solutions need to be light weight means minimal assumption about engineering environment into which it deploy and simple enough that can be adopted in practice. Traditional approaches based on the assumption that all requirements are clear before implementation. In practice customers are not clear with all requirements and thus not software manager at early stage. To satisfy the customers, the software must meet current as well as previously stated all requirements. The question today is not how to stop changing nature of requirements but how to handle the changing nature of requirements through the life cycle within time and budget. The paper [14] addresses key challenges viz. how compositional effect component, how to extract non-functional requirements in early stage, how evolving requirements impact can be reduced, how the shifting of traditional to new model affects the development of the system, how to provide user flexibility, how to make the system more domain-independent in software engineering.

The main reason behind evolving nature of software requirements is the diversity of stakeholders [15]. The resulting method is light weight, agile methods where developer team produces the first delivery in week, to achieve an early win and rapid feedback. It invents simple solutions, so there is less to change and making those changes easier; improve the design quality continually, making the next version less costly to implement; new technology and new business rules [16]. In agile process individual expertise and quality is required. Furthermore, it needs very close relationship with business expertise. This method fails in the lack of frequent communication with involving people including all teams, customers and business expertise. Various agile models viz. Extreme Programming (XP), Scrum, Cockburn's Crystal Family, Dynamic System Development Method (DSDM), Feature Driven Development (FDD), and Adaptive Software Development (ASD) are available. Paper [17] presents a brief detail on each. While traditional approaches are designed to deal with the large project, stable requirements, and complete documentation, agile methods are developed with intention specifically to cope up with the changing nature of requirements. Agile models start early coding to identify early flaws in the system and by which improve the system at an early stage. Thus, it is adaptive in nature.

Table 2. Comparative analysis of Agile models

Features	**Extreme Programming (XP)**	**Scrum**	**Cockburn's Crystal Family**	**Dynamic System Development Method**	**Feature Driven Development**	**Adaptive Software Development**
Work flow	Incremental	Incremental + + Short	Incremental +	Incremental +	Incremental +	Incremental + Short

		Iteration	Short Iteration	Short Iteration	Short Iteration	Iteration
Development Approach	Adaptive	Adaptive	Self-adaptive	Adaptive	Adaptive	Self-adaptive
RS: nature	Evolving	Evolving	Evolving	Evolving	Evolving	Evolving
Project size/type	Small	Small/ Large	Small/ Large	Small/Large	Large	Small
Focus	Delivery according user needs/technical leadership	Delivery according system developer/management	Delivery according user needs/ important component deliver first -Review	Delivery according user needs/ training	Delivery according user needs and feature list	Delivery according user needs
Documentation	Less	Less	Less	Moderate	High	Less
Communication	Informal meeting	Informal meeting	Standard meeting	Standard meeting	Document sharing	Document sharing
Changes in sprint	Allowed	Not allowed	Allowed	Allowed	Allowed	Allowed
Iteration time	Weekly	Two week or one month	Within four moths	Quickly	Less than 2 week	Four to five weeks
Discipline process	High	Moderate	Less	Moderate	Moderate	Moderate

All agile models support common characteristics viz. evaluation after each iteration, produce early prototype, simplicity, flexibility, easy risk management, need highly experienced development team, less process time compare to the traditional models, client satisfaction, frequent communication/feedback and quick delivery [19, 20, 21, 22]. Apart from these there are some distinct characteristics of each agile model that are listed in table 2. However, it has downsides viz. no formal documentation before code, requires extremely skilled development team, overlapping phases. Furthermore, it is not feasible to start coding before proper requirements analysis. The major question is how to divide the whole system into small working iterations if requirements are not analyzed or with less documentation? From table 1 and 2 one can conclude that there are drastically change in traditional and agile models. Both have their positive side as well negative. Applicability of traditional and agile models in a practical environment is always leading to confusions. There are many factors such as listed in table 1 and 2 that helps to choose the appropriate one. Though, the literature shows the existence in the follower's who favor traditional model and others who favor agile model. There exists a class in the literature that favor the combination of both traditional and agile models by which one can benefit from the advantage of both models called Hybrid model [23]. The major pitfall of traditional models is less communication that is the biggest advantage of the agile model. In the hybrid model at initial stage the traditional models are used for documentation and design purpose, then the agile model is used to make the coding and testing considering frequent communication with customers means the continuous feedback analysis and weekly product release. The hybrid model can be any combination of traditional models and agile models depends on software environment and software manager. There are different criteria such as the size of the project, evolution in requirements, duration of the project, criticality based on that we can select the combination of available SDLC models as a hybrid model. The major issues in the traditional models are to achieve predictable schedule and evolving requirements. To create software is a creative task that cannot be predictive in nature so we need to follow the model that support adaptive nature and cannot plan in advance. If the dedicated and experienced team is there and continuous customer involvement and the time line is short then one can go with the agile models also termed as light weight, code oriented, adaptive, people oriented. Finally, to achieve the benefit of both traditional and agile one can go with the hybrid models. In the traditional methodology, the deviation is considered as mistakes that one should correct while in agile methodology it considered as guidelines towards a correct solution. The agile model keep learning at each iteration and more realistic compared to the traditional approaches. Furthermore, the agile methods follow RE throughout the development life cycle in small, informal stages based on constant feedback from the various stakeholders. Though the agile RE practice has positive impacts viz. frequent communication between users, dynamic and adaptive nature it has a downside too [24, 25]. Table3. summarizes positive and negative aspects of the agile RE practice compare to the traditional models [2, 10, 7]. The major downside using agile method is a lack of focus on non-functional requirements because users focus mainly on what the system will do (functional requirements) and not regarding quality viz. maintainability, performance, and safety [11]. In fact the non-functional requirements identified at the time of development that make the scenario worse.

Table 3. Positive and Negative impact of Agile RE Practic

Agile RE practice	Traditional RE practice	Positive Aspects	Negative Aspects
Face-to-face communication(user stories)	Written specifications/large documentation	-Customers play an active role in evolving requirements -Informal communication prevent the need for time-consuming and large documentation	-Effectiveness depends heavily on quality interaction between users. -Lack of frequent communication can lead to high risk
Iterative requirements	Quality and Stability concerns in requirements	-Clear interpretation of requirements due to frequent communication/ communication when needed.	-Difficult to estimate cost and schedule as evolving requirements. (features are not fixed prior)
Requirement prioritization at each phase	-Prioritized once -Many factors affect.(cost, risk, implementation)	-Better meet customer's priority needs -Provides numerous opportunities for re prioritization	-Quality requirements may neglect at initial stage by customers that are essential for success of any software. -Instability
Managing requirements change	Early but not constant	-Early and constant validation of requirements minimizes the need for major changes	-Increases project cost
Prototyping	Formal Requirements Specification	-Obtain quick customer feedback on requirements	-Difficult to maintain evolving prototype -Create unrealistic expectations
Use review meetings and acceptance tests	Early phase user communication	-Provide early progress reports to the users to review the developed features and get feedback -Achieve users trust -Help to obtain management support	-No formal specification modeling of requirements

CONCLUSIONS

The software development life cycle provides a baseline to develop a software. The analysis of the three broad categories of models viz. traditional model, agile model, and hybrid model answers a difficult question in practice how to decide which model is suitable? We can conclude that if requirements are stable, lack of dedicated expert team, need of large documentation and the timeline is more than one can go with traditional waterfall model also known as predictive model, process oriented (or heavy weight due to large documentation and planning). To prove the benefits of agile models over traditional models and the same way benefits of hybrid models over Agile models seems indecipherable because they suit best to some environment. This study shows the applicability and success of models vary based on factors viz. nature of requirements, project team and its expertise, project size, project cost and its duration, delivery time etc. Furthermore, in practice it helps to select an appropriate and effective model for software development that reduce the cost and reach the time limit. The major problems faced by three categories are unmanaged evolving requirements, lack of healthy user communication and insufficient domain knowledge. In practice developers moving to adopt the hybrid model to achieve the key benefits of traditional and agile models. However, applicability still depends on the software environment.

REFERENCES

[1] Davis, Alan M., Edward H. Bersoff, and Edward R. Comer. 1988. A strategy for comparing alternative software development life cycle models.Software Engineering, IEEE Transactions on14.10: 1453-1461.

[2] I. Sommerville. 1996. Software Engineering. Fifth Edition, New York: Addison Wesley.

[3] Jacobson, I., Booch, G., Rumbaugh, J., Rumbaugh, J., & Booch, G. 1999. The unified software development process (Vol. 1). Reading: Addison-wesley.

[4] Ghezzi, C., Jazayeri, M., & Mandrioli, D. 2002. Fundamentals of software engineering. Prentice Hall PTR.

[5] Petersen, K., Wohlin, C., & Baca, D. 2009. The waterfall model in large-scale development. In Product-focused software process improvement (pp. 386-400). Springer Berlin Heidelberg.

[6] Ruparelia, N. B. 2010. Software development lifecycle models. ACM SIGSOFT Software Engineering Notes,35(3), 8-13.

[7] Kleijnen, J. P. 1995. Verification and validation of simulation models.European Journal of Operational Research, 82(1), 145-162.

[8] Goel, A. K. 1991, June. Model Revision: A Theory of Incremental Model Learning. In ML (pp. 605-609).

[9] Geambasu and I. Jianu. 2011. Influence factors for the Choice of a Software Development Methodology. Accounting and management Information Systems, Vol. 10, No.4, pp. 479- 494, 2011.

[10] Larman, C., & Basili, V. R. 2003. Iterative and incremental development: A brief history.Computer, (6), 47-56.

[11] May, E. L., & Zimmer, B. A. 1996. The evolutionary development model for software.Hewlett Packard Journal,47, 39-41.

[12] Naumann, J. D., & Jenkins, A. M. 1982. Prototyping: the new paradigm for systems development.Mis Quarterly, 29-44.

[13] Boehm, Barry W. 1986. A spiral model of software development and enhancement. In ACM SigSoft Software Engineering Notes, Vol. II, No. 4, 1986, pp 22-42.

[14] Finkelsteiin, A., & Kramer, J. 2000, May. Software engineering: a roadmap. InProceedings of the Conference on the Future of Software Engineering (pp. 3-22). ACM.

[15] Fuggetta, A. 2000, May. Software process: a roadmap. InProceedings of the Conference on the Future of Software Engineering(pp. 25-34). ACM.

[16] Highsmith, J., & Cockburn, A. 2001. Agile software development: The business of innovation.Computer,34(9), 120-127.

[17] Matharu, G. S., Mishra, A., Singh, H., & Upadhyay, P. 2015. Empirical study of agile software development methodologies: A comparative analysis.ACM SIGSOFT Software Engineering Notes,40(1), 1-6.

[18] Fowler, M. 2001. The new methodology.Wuhan University Journal of Natural Sciences, 6(1-2), 12-24.

[19] Dybå, T., & Dingsøyr, T. 2008. Empirical studies of agile software development: A systematic review. Information and software technology, 50(9), 833-859.

[20] Paetsch, F., Eberlein, A., & Maurer, F. 2003, June. Requirements engineering and agile software development. In null (p. 308). IEEE.

[21] K Schwaber, "Scrum Development Process. 1995. Presented at OOPSLA'95 Workshop on Business Object Design and Implementation, 1995.

[22] J. Stapleton. 1997. Dynamic Systems Developments Method - The Method in Practice: Addison Wesley.

[23] Juyun Cho. 2009. A Hybrid Software Development Method for Large-Scale Projects: Rational Unified Process with Scrum. Issue in Information Systems. Vol. 10, No.2, pp. 340-348.

[24] Cao, L., & Ramesh, B. 2008. Agile requirements engineering practices: An empirical study. Software, IEEE, 25(1), 60-67.

[25] Grau, R. 2012. Requirements engineering in agile software development. InSoftware for People (pp. 97-119). Springer Berlin Heidelberg.