REVIEW OF THE MYTHS ON ORIGINAL SOFTWARE DEVELOPMENT MODEL

Sriramasundararajan Rajagopalan[1]

[1]Agile Training Champions

ABSTRACT

Software development is integral to today's digitally monopolized business environments with increasing mobile, web, and desktop applications. With the growing emphasis to accommodating change in software development using agile approaches, the software development life cycle (SDLC) is often equated synonymously with the waterfall approach that never existed in the original proposition of software development life cycle. This paper demystifies these concepts so that the SDLC is correctly understood further emphasizing that similar misinterpretations do not lead to incorrect understanding of agile methodologies during software development.

KEYWORDS

SDLC, Waterfall, Software development life cycle, SDLC, traditional development, V-Model, Iterative Prototyping, Incremental Development, Agile, Lean, Scrum, XP, Kanban, ROI

1. INTRODUCTION

Long gone are the days where one can avoid thinking of software development without incorporating servicing the customers with the software developed. The exponential growth of mobile commerce has led to an abundance of software to the extent that the software itself is no longer reserved for Information Technology (IT) industry alone. Whether the software developed is for the customers external to an organization or the employees and vendors internal to an organization, the software development life cycle (SDLC) needs to carefully build good quality software and also rollout and maintain the software released.

This trend led to the proposition of the SDLC by Winston Royce [1] containing a series of phases from the conception of an idea or challenge to be addressed through large software development. This SDLC model evaluated the return on investment (ROI) in developing the required software, the actual design and development needs, the acceptance of the software through testing, and the post-development stages of deployment and maintenance. The widespread adoption of SDLC started when the federal mandate [2, 3] adopted this model as DOD-STD-2167A labelling it as waterfall. Consequently, Royce's original foundation morphed as practitioners interpreted it wrongly leading to SDLC being a linear model.

In fact, Royce [1] was a building a software development methodology incrementally using ten diagrams where each diagram was iteratively adding features to the model suggesting best practices to employ and highlight the risks when these guidelines are not met. The most popularized linear model [4] with a series of seven steps was the second diagram as a precursor to software development discussion, when the software developed is going to be used by people other than the developers that built it. However, that diagram was perceived to be the actual SDLC model. With the absence of the feedback loop in this second diagram in Royce story building approach to SDLC [1], the practitioners drew parallel to water falling from a higher plane to a lower plane attributing the waterfall approach to software development as the original SDLC model. Nevertheless, nowhere in Royce's model [1] is present any reference to the waterfall terminology. In this paper, many such practitioners' interpretations convoluting original theoretical foundations are demystified so that similar misinterpretations do not bleed into the agile thinking or organizations can take inventory of the right best practices even in their current software development processes.

2. MYTHS AND FACTS OF SOFTWARE DEVELOPMENT PROPOSITION

The literature is rich with several studies describing the history of software development [5, 6, 7]. Royce sketched his thoughts of software development in 1970 laying the foundations of the SDLC with various interconnected stages. Contrary to the claims that this SDLC is a linear waterfall design [8], Royce's detailed notes [1] define an iterative and cyclical approach to software development. For instance, Royce [1] called this linear model will lead to failure (p. 329) when the customer is not involved at earlier points. However, even the agile manifesto claiming the importance of the customer collaboration needs in software development claim that the original proposition lacked customer collaboration until the final delivery.

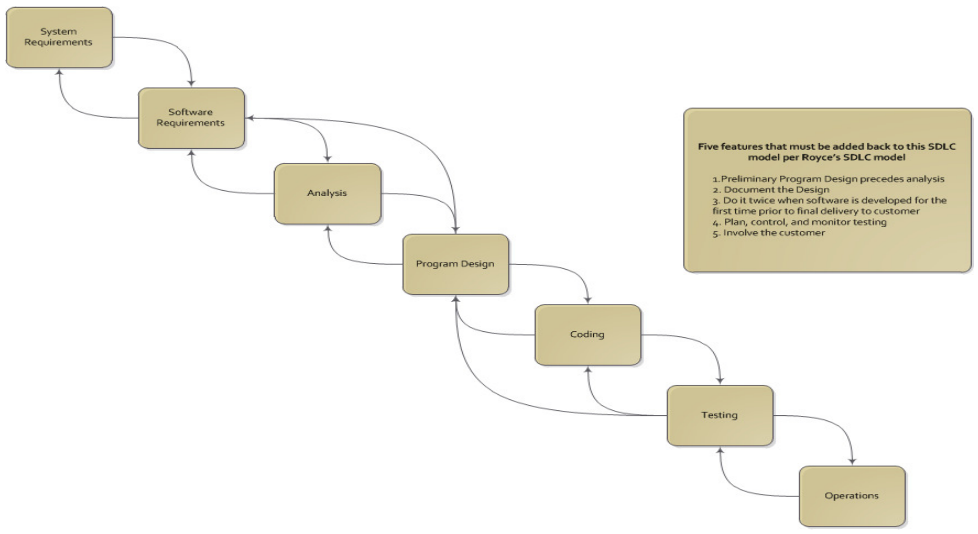
It is therefore evident that there are several misinterpretations of the original model that is now classified as "traditional" or "waterfall". In fact the popularity of these incorrect associations has become so mundane that the waterfall model of "river of no return" [8] has become synonymous with the original SDLC. As a result, the approaches that emanated from the specialists in the information technology (IT) arena leading to the agile approaches are perceived as significant deviations from the SDLC model due to the misinterpreted claims, such as the following. Labelling Royce's model [1] as the original baseline model moving forward in this paper, these claims against SDLC are challenged.

1. Linear approach to software development with no feedback
2. Big upfront requirements gathering
3. Gathering requirements upfront saves cost
4. Analysis follows requirements followed by design
5. Project Management is not part of software development
6. High degree of documentation needed before starting work
7. Customer sees work after all the work is developed and tested
8. Testers need not be involved early

2.1. Myth 1: Linear approach

As noted earlier, several claims [5, 6, 7, 8] note that the Royce's SDLC model [1] is made up of a series of sequential and linear stages. These include the system requirements, software requirements, analysis, program design, coding, testing, and operations where the output of each stages feeds the next stage. Royce [1] set the stage by developing a series of stages as the first step when the software developed is going to be used by more than the persons that built it and expanded on this initial proposition of seven sequential stages as "...but the implementation described above is risky and invites failure." (p. 329)

Royce's model [1] expanded on these risks noting that several of the performance testing requirements and storage limitations may be difficult to gather completely at the requirements phase and will differ during development and testing phases. Practical insights on the differences among the production, test, and development environments, such as the CPU, memory, cache, and network transfer are not always consistent for linear approach to early gathering of requirements feeding the design and development for testing to succeed flawlessly. As a result, Royce [1] suggested including two types of feedback mechanism in the model.



Five features that must be added back to this SDLC model per Royce's SDLC model

1.Preliminary Program Design precedes analysis
2. Document the Design
3. Do it twice when software is developed for the first time prior to final delivery to customer
4. Plan, control, and monitor testing
5. Involve the customer

Figure 1. Feedback loops in original SDLC model

The first feedback loop included responses from every stage to its preceding stage as illustrated in the Figure 1. Royce [1] noted that while testing can unearth bugs in development and development can suggest program redesign, there is a need for the findings from testing to actually be attributed to the program design and the program design due to the inaccurate requirements gathering emphasizing the iterative relationship in software development. Therefore, Royce's recommendation [1] included a skip level feedback to a stage before the previous stage demonstrated in Figure 1 in addition to feedback to the previous stage. These discussions in Royce's original model [1] eliminate the reference to the current thinking that the SDLC model was linear without feedback. Therefore, the fact is that the waterfall approach never existed in theory and only existed in the incorrect implementations.

2.2. Myth 2: Big upfront requirements gathering

The popularity of the agile approaches to software development is their focus on welcoming changes [9] with high degree of uncertainty [10] even if the changes are introduced late in the project and impacting even the architectural design [11]. Inherent in the earlier claim of linear approach to software development was another hidden notion that Royce's SDLC model [1] required gathering all the requirements from the user prior to beginning the analysis stage leading to the technical, operational, and environmental feasibility studies.

Royce's original stand [1] is truly agile in nature when proposing "... change process is scoped down to manageable limits" (p. 328) requiring a firm signoff on requirements gathering. Royce however didn't timebox this phase leading to the teams taking more time for gathering as much as they can. The agile approaches differ here because they enforce a time limit [9] on this phase so that there is consistent cadence in feature delivery in the incremental software updates. This approach to signoff is no different in the agile approach where the iteration planning focuses on what user stories can be accommodated within an iteration based on the team's committable velocity. Royce [1] states that this closure to requirements allows maximizing the extent of early work. The lean concepts of limiting the work in progress espoused by Kanban and disallowing changes in iteration as noted in Scrum [9] are integrated in the original SDLC proposition so that the software can be designed appropriately. These observations invalidate the claims that the SDLC proposition required gathering all the requirements prior to design and development.

Additionally, Royce [1] even questions if all system requirements can be gathered at the requirement and analysis stage while observing that the later events [testing] is the first entry to gather data points on "timing, storage, I/O transfer" leading to the introduction of "disruptive changes in design" in emphasizing skip level feedback in refining requirements later and updating the build. Therefore, the claim that the SDLC required all upfront requirement gathering is not a fact. However, those that conceived the waterfall mindset to software development ended up with band-aid fixes to the code leaving the requirements not comply with the code deployed violating the original requirements of software development.

2.3. Myth 3: Gathering requirements upfront saves costs

One of the underlying premises to upfront requirements is that gathering the requirements early saves costs as it helps plan the project better by eliminating rework. Regardless of the software development methodology, planning is essential but the plan itself is not because changes are not completely known, predictable, or constant. Neither the agile approaches to software development [9] nor Royce's original model [1] as mentioned earlier prohibits entry of changes in software development. But, did Royce promise cost savings by adopting this model?

Certainly, Royce [1] takes a definitive stand that when changes are introduced, one can expect higher percentage of schedule and cost overrun. (p. 329) Royce differentiates the minor deviations from missed steps in requirements or inefficiencies in design that could be addressed by small code changes from disruptive design changes that may lead to major changes in design and development. When such disruptive changes become necessary, either due to poor software design or due to forces external to the team's original understanding, then Royce magnifies the cost overrun and schedule slips. Neither does agile approaches promise cost reduction in software development as they embrace changes [9, 12] and the state of agile development survey [13] conducted by VersionOne does not include cost savings as one of the benefits seen by organizations in using agile (p. 10).

There is an important distinction to call out between Royce's SDLC model [1] and the agile approaches to accommodating change. Royce's original position that minor changes in the code can fix design flaws with no disruptive feedback into the other areas. This observation is not entirely true because these flaws when unchecked can lead to technical debt where flaws and bugs are knowingly passed into design and development. Agile approaches significantly differ here [9, 12] by introducing a set of processes and tools, such as refactoring, test driven development, continuous improvement, and pair programming to contain technical debt.

2.4. Myth 4: Design follows Analysis

The literature is profuse with both the scholars and the practitioners stating that the SDLC model uses detailed analysis first before design is executed. Royce [1] proves this claim wrong in one of the five additional recommended features that will ensure success of the SDLC model laid out in Figure 1. The first of these five features is introducing a program design phase before the analysis. Royce reasons that the time spent on this program design may help gather failsafe requirements for the software to function better despite the ambiguous requirements. (p. 331)

Royce [1] also recommended that the skills in the program designers need to encompass a number of domains like database design, application interfaces, and operating procedures even at the risk of being wrong bringing early signs of kaizen and continuous improvement so that these designs can be refined through iterations. According to Royce, this program design concludes with a system overview document that at least one person has a deep understanding with. During the period when this model was proposed, most large scale implementations were predominantly in mainframe but its implications in the current mobile, distributed, and client based application development requirements call for merged roles like business systems analysts who can think in terms of the high level business requirements and simultaneously converse with developers and network architects.

These findings are indispensable in the evolution of the rapid application development (RAD) to develop a prototype as a proof of concept and the development of spikes in the agile context to validate the concepts prior to committing time and resources. The surge of object oriented analysis and design (OOAD) and the unified modelling language (UML) were adequately supplemented by integrated development environments (IDE) where graphical elements can be quickly assembled with minimal coding to support the needs of these hybrid roles. Although these program designs were later followed by the detailed design, these findings invalidate the claims that Royce's original SDLC model [1] was linear with the analysis preceding design.

2.5. Myth 5: Project Manager skillset in software development

Perhaps due to the reference to the word "software" in the software development life cycle, the emphasis of strong project management in managing the IT projects is losing its focus leading to repeated claims on IT failures due to project management failure [14, 15, 16, 17]. The project management book of knowledge (2014) provides a number of tools and techniques for the project manager [18] to avoid such failure.

Interestingly, Royce [1] took a firm stand on project management in software development suggesting very strongly to "replace project management" (p. 332) in the second recommended feature when the documentation of the software requirements go missing. Royce rationalized that without a clear document, the project management cannot confirm any tangible evidence of task completion succumbing to the "90% finished" syndrome. Although the agile approaches may make the requirements documentation simpler in the form of user stories, the definition of done is baked into documentation expectations in Royce's proposition for good project management as success criteria in quality software development.

2.6. Myth 6: High degree of documentation is needed

In the degree of documentation, Royce's proposition [1] is unequivocal. Although the claims to having big upfront gathering of requirements document is invalidated by earlier comments, Royce mandates six types of documentation that needs to be updated along with the final delivery of the software product. These six documents include initial software requirements, preliminary design document, interface design document, final design document, test plan document, and operating instructions.

The agile manifesto carefully suggests working software over comprehensive documentation but didn't abolish documentation. Many situations in the agile context require additional documentation besides just the user stories. For example, when the team structure uses the distributed team structure through computer supported collaboration tools or involves the regulated industries like the pharmaceutical or financial industry requiring compliance documents to meet PhRMA or Sarbanes-Oxley mandates, or when the service provider relationships with partners call out Capability Maturity Model Integration (CMMI) compliant requirements, additional documentation is called for [9].

Royce [1] emphasizes that until development begins coding, the documentation, specification, and design denote the same thing (p. 332) where the phases can run concurrently to elicit requirements and validate them through the design. On the contrary, the documentation takes on special significance in the testing phase where testing is mapped to the requirements to identify errors in design and development as pointed out by skip level feedback and also in the operational phase so that the software can be maintained and managed by members other than those that developed the software.

2.7. Myth 7: Customer sees work after all requirements are developed and tested

The misconceptions of a linear model and big upfront requirements gathering made Royce's original SDLC model [1] as a black box where there was no user involvement between requirements gathering and final software delivery. These observations are so widely publicized in the several renditions of images in the cyberspace where the customer's requirements were so vastly different from what was finally delivered. Yet, these observations of delaying the user or the customer until later stages are unsubstantiated in Royce's model.

On the contrary, in the fifth feature recommendation, Royce [1] stresses the importance of involving the customer in a formal way at earlier stages before the final delivery further underscoring the trouble when such early customer involvement is absent after requirements gathering. (p. 335). Additionally, Royce promotes a success criterion as the third feature requirement that prior to the final delivery the customer should receive the software for evaluation [fitness for use] and the entire delivery process repeated in mini-scale to address the issues from this evaluation. This testing is formalized in project management and agile context requiring the business users to perform acceptance testing sowing the seeds for acceptance test driven development (ATDD) [9] prior to the final delivery to the customer. Yet, these acceptance testing by business users and customers prior to final delivery are less frequently implemented leading to escaped quality defects, project failure, and customer dissatisfaction

2.8. Myth 8: Testers need not be involved early

If work moved from one group to another group sequentially, then, the testers would be involved only after development is complete. Royce [1] never took this stand to begin with and recommended several levels of testing that needed different types of specialists during design and development. Royce challenges the notion of designers understanding design and developers relating to coding as a "sure sign of failure" emphasizing that different specialists need to be involved in testing the design and development against the documented requirements. According to Royce, testers bring a special set of skills and should be involved early to review design decisions and suggest recommendations before development begins.

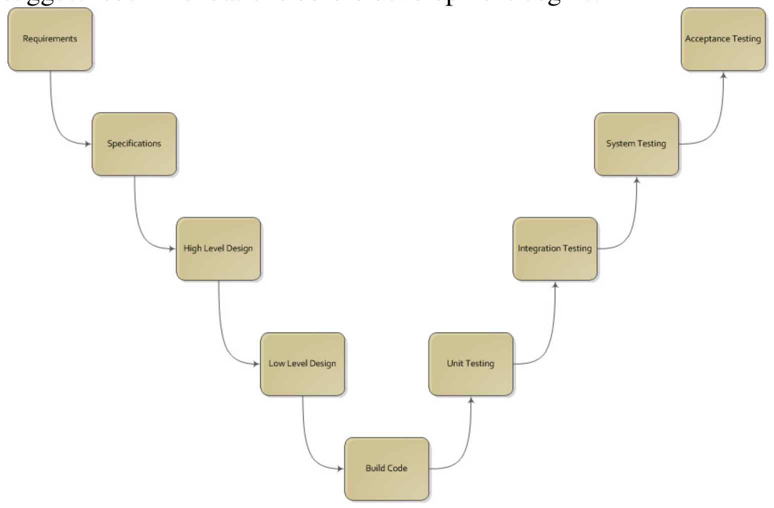


Figure 2. V-Model establishing testing alignment

Royce [1] recommended visual inspection of design and development by a second party who neither contributed to design or development suggesting program review and formalized code review protocols that should be a prerequisite to good software development. Royce insists that the computer shouldn't be relied on such testing as it could be expensive. At the time of Royce proposition, the advanced compiler design, integrated programming environment, advanced programming languages or the tools for continuous build & integration, automated testing, and workflow tools were unavailable and so this claim is worth the review for computer based testing.

For instance, the compiler design today are advanced to even track the garbage tracking automatically and the integrated development environment could even offer suggestions on uninitialized variables at the time of coding or detect potential memory leaks improving security and stability in the development. The V-Model to software development [19] involving validation and verification model took on these principles where parallel level testing was mandated as demonstrated in Figure 2. Agile development also involves the testing specialists to get involved to get engaged in conversations about design and development when the requirements are still evolving [9, 20].

2.9. Discussion

There is a saying, "In theory, there is no difference between theory and practice. In practice, there is." But, if practice created a theory that didn't exist, then, the theory needs to be re-established to shed light on these findings. Theory should augment practice and practice should reinforce theory and when this cycle is broken, this is a sure sign of chaos. Software development has evolved with a number of developments in the computer aided software engineering tools. Yet, various scholar-practitioner discussions revolve around waterfall approaches to software development that was never proposed in the original SDLC model. While there has been refinements made with spiral approaches, V-Model, and agile approaches [9, 19, 20], misinterpretations in these refinements and new ideas could lead to additional challenges unless corrected.

3. CONCLUSIONS

The practitioners' rush interpreting the foundation of SDLC proposition incorrectly can be seen as one cause for creating a theory of waterfall that never was proposed. Similarly, the scholar's stand believing the voice from the corporate setting to be the outcomes of the theory's weakness mutated the original proposition. Challenging these observations, Royce's model [1] raises the risks with these incorrect implementations as suggested below. As agile approaches to software product development gains popularity, it would be a worthwhile effort for organizations to re-evaluate the software development practices and apply the proper criteria to use the effective software development methodology.

1. Linear approach to software development with no feedback is a risk
2. Big upfront requirements gathering is not always possible
3. Gathering requirements upfront is not guaranteed to save cost or reduce timeline
4. Program design should be incorporated prior to full analysis
5. Project Management is an integral part of software development
6. High degree of documentation needed before final delivery but documentation should always exist
7. Customer should be involved frequently for software development success
8. Testers need to be involved earlier in the stages prior to development

ACKNOWLEDGEMENTS

The author would like to thank all the customers, team members, and stakeholders involved in all the projects the author had managed and the learners the author has trained in both the corporate and academic settings from who the author unlearned and relearned software development life cycle.

REFERENCES

[1] Royce, W. Winston (1970) "Managing the development of large software systems", Proceedings, IEEE WESCON, August, pp 1-9.
[2] Boehm, B. (1996). Anchoring the software process. IEEE Software, July, pp 73-82.
[3] Gabig, J.S. (1991). Managing software development: An insight into process. National Contract Management Journal, 24, 2, pp 41-50.
[4] Palvia, P., & Nosek, J. T. (1990). An empirical evaluation of system development methodologies. In Managing information resources in the 1990s: proceedings of 1990 Information Resources Management Association international conference (p. 72).
[5] Misra, S. (2012). Agile software development practices: Evolution, principles, and criticisms. The International Journal of Quality & Reliability Management, 29, 9, pp 972-980.
[6] Gremillion, L.L., & Pyburn, P. (1983). Breaking the system development bottleneck." Harvard Business Review, March.
[7] Guimares, T. (1985). A study of application program development techniques. Communications of the ACM, 28, 5. Pp 494-499.
[8] Tayntor, C.B. (2007). Six sigma software development. Boca Raton, FL: Auerbach Publications.
[9] Cohn, M. (2010). Succeeding with agile. Upper Saddle River, NJ: Addison Wesley.
[10] DeCarlo, D. (2004). eXtreme Project Management. San Francisco, CA: Jossey-Boss.
[11] Crow, A. (2012). The PMI-ACP exam. Velociteach Publications.
[12] Saddington, P. (2013). The agile pocket guide. Hoboken, NJ: John Wiley-Sons.
[13] VersionOne (2014). 7th annual state of agile development survey, pp 1-14.
[14] Betts, M. (2003). Why IT Project Fail. Computerworld, Volume 37, Issue 34, Page 44.

[15] Fichter, D. (2003) Why Web Projects Fail Volume 27, Issue 4, page 43.
[16] Auerbach, B. & McCarthy, R. (2014). Does Agile + Lean = Effective: An investigative study. Journal of Computer Science and Information Technology, 2, 2, pp 73-86.
[17] Keith, M., Demirkan, H., & Goul, M. (2013). Service oriented methodology for systems development. Journal of Management Information Systems, 30, 1, pp 227-259.
[18] A guide to the project management book of knowledge (2014). Pennsylvania, PA: Project Management Institute.
[19] Balaji, S. & Murugaiyan, M.S. (2012). Watefall Vs V-Model Vs Agile: A comparative study on SDLC, International Journal of Information Technology and Business Management, 2, 1, pp 26-29.
[20] Leau, Y.B., Loo, W.K., Tham, Y.P, & Tan, S.F. (2012). International Conferences on Information and Network Technology, 37, pp 162-167.
[21] Vickers, M.H. (1999). Information technology development methodologies: Towards a non-positivist development paradigm. Journal of Management Development, 18, 3, pp 255-272.

Authors

Dr. Rajagopalan graduated with a Bachelor's degree in Electronics and Communication Engineering from the University of Madras in India, a Master's (MS) degree in Computer Engineering from Wayne State University, Michigan, a business (MBA) degree in Management from Concordia University, Wisconsin, and a doctorate (PhD) degree in Organization and Management from Capella University, Minnesota. Dr. Rajagopalan also holds many professional certifications in project management (PMP, PMI-ACP, PMI-SP, CSP, CSPO, CSD, CSM, ACC, IT Project+). With extensive software development and project management experience in many industries, Dr. Rajagopalan promotes the scholar-practitioner approach delivering information technology and management courses as adjunct faculty in several universities and community colleges in addition to delivering his own project and agile trainings.