Software Development Life Cycles: History and Future

Aden Kenny

Abstract

This paper discusses the importance of the use of process models in the software development life cycle, provides a review of the process models used so far, compares and contrasts five different process models and then attempts to discuss the future of process models.

1 Introduction

The Software Crisis was a time period in the infancy of software development where it was found to be difficult to write software that was useful and efficient on the hardware of the time. This led to a large quantity of poor quality software being created that did not meet the requirements or was not even delivered at all.

It has been stated that there were problems of "achieving sufficient reliability" and difficulties of meeting time requirements and specifications on large scale projects amongst other problems that were discussed at the *NATO Software Engineering Conference, 1968* (Naur and Randell, 1969).

As a result of this conference where major problems with the state of software development were brought up, software development process models started to become far more popular as they were seen as a major part of the solution to the *Software Crisis*.

2 The importance of process models

During the *Software Crisis* discussed above, many problems were faced possibly to the lack of direction in the development of software. These issues were things like projects being over budget, projects running over-time, inefficiency in the delivered software, projects that did not meet the set requirements, unmanageable projects, and projects that never delivered software.

A solution that was proposed to the problems that software was facing was the idea of a development process guided by the usage of a framework. These frameworks later became known as 'process models'.

Nearly all software development process models start with a requirements gathering phase where the requirements that need to be met are gathered. This step is extremely important to the development of software as a program is highly unlikely to meet the requirements set by the client if the requirements have not been formally gathered and analysed.

A process model proves itself extremely useful in large scale, team projects. This is because a process model forms a base on which a team can find a common understanding in the development of software. This is due to that idea that all software development projects have a process called the *software life cycle* . This involves a number of steps starting with "Requirements analysis and definition", "System design", "Program design", "Writing programs (implementation)", "Testing", and "System delivery (deployment)" (Ma, 2017).

A set process model in a project means that a team will have a way of fulfilling the process of the software life cycle. This becomes important as it allows the team to have a common understanding what needs to be done at all times no matter what model is used.

3 Review of proposed models

One of the first formally described process models was the the *Waterfall model* by Winston Royce in 1970. Royce states that he believes in the concept of analysis before coding but presents the model as "risky" and that it "invites failure" (Royce, 1970).

The main idea to take away from Royce's paper is that analysis is a required step in the software development life cycle. This is very important as good software cannot be developed without fully understanding the problem that the software is designed to solve and all of the complexities involved, of which there will inevitably be many.

The waterfall model has been criticised as it is quite inflexible in some situations especially in the case of changing or unclear requirements. If a new requirement is requested or found to be needed it is not possible to go back a phase and insert this new requirement. The new requirement must either be ignored and not implemented, or the entire process must be restarted. Other problems include a long development time with nothing to show before the end and the idea that it "Views software development as [a] manufacturing process rather than a creative process" (Ma, 2017).

It was realised by some people quite soon after the adoption of the waterfall method that it was flawed in some situations such as by Royce where he first formally describes it and criticises it (Royce, 1970). This led to the development of other process models that were designed to fix the shortcomings of the waterfall model. Examples of these models include *Waterfall model with prototyping.*

This is an arguably more flexible version of the

waterfall model. It involves following the same design cycle as the waterfall model but involves prototyping in the design phase and the ability to move back to the requirements and system design phases in the testing phase.

The idea of a more flexible process model than waterfall has been frequently seen as advantageous in many situations which has recently led to more 'lightweight methods' gaining popularity. These lightweight methods generally focus more on flexibility and the opportunity to go back to previous cycles to fix mistakes and add features that were not in the initial requirements, either through changing customer requirements or incomplete requirements gathering.

The waterfall model has been described as a "heavyweight method" (Ma, 2017). These 'heavyweight methods' do have advantages over so called 'lightweight models'. The waterfall model provides an extremely useful case to examine the advantages of heavyweight models as it is possibly the most frequently used heavyweight model and because of its long history and the large amount of use it has had.

One of the main cited advantages of the waterfall model is it is frequently possible that errors in design are found before any code is actually written (Hughey, 2009). This saves time during the implementation phase as there will be no need, in an ideal situation, to go back to the planning stage of the life cycle. An additional frequently mentioned advantage is the idea that a very structured model means that it is far easier to measure the progress of the project when compared to more lightweight models, as the project milestones are clearly defined.

A major difference between the methodologies that it is frequently cited is the idea of iterative development. More lightweight, agile, methods such as Scrum or Extreme Programming (XP) prefer that features are developed iteratively and favours short release cycles with frequent feedback from the customer. This is contrast to so called 'less flexible' models like waterfall. These more heavyweight models call for a single release at the end. Critics of waterfall state that this can be harmful to the project as it becomes extremely infeasible to get customer or stakeholder feedback on the current state of the project.

References

Naur, P. & Randell, B. (1969), "Software Engineering - Report on a conference sponsored by the NATO Science Committee".

Royce, W. (1970), "Managing the Development of Large Software Systems" *in* Proceedings of IEEE WESCON.

Ma, H. (2017), "Classical Software Life Cycle Models" *in* lecture notes distributed in SWEN301.

Larman, C. & Basili, V. (2003), "Iterative and Incremental Development: A Brief History" *in* IEEE Computer (June ed.). 36: 47-56.

Hughey, D. (2009), "Comparing Traditional Systems Analysis and Design with Agile Methodologies". University of Missouri - St. Louis.