A Pipelined Approach for Iterative Software Process Model

Ms.Prasanthi E R, Ms.Aparna Rathi, Ms.Vardhani J P, Mr.Vivek Krishna

Electronics and Radar Development Establishment

C V Raman Nagar, Bangalore-560093, INDIA

Tel No. 080-25025769 Fax No. 080-25240821

prasanthi_e_r@yahoo.com

Abstract:

In complex scientific projects where the whole system cannot be completely perceived in the beginning, an iterative development approach that follows the functionality to be delivered in parts has become a necessity and an effective way for refinement and risk management. Iterative development in conjunction with software reuse is one of the most promising and practical ways of tackling risks involved in a complex system. In the iterative model the feedback from the stage wise testing at the earliest can be used to improve the final deliverable. In this paper we share our experience with iterative software development in which multiple iterations are carried out in a parallel fashion. Pipelining concepts are employed to have multiple iterations executing concurrently leading to a reduction in delivery time and early diagnosis of risks. We illustrate the use of this process model through an example of an airborne radar application software project.

This paper is organized as follows. It starts with introduction dealing with a comparison between waterfall model and iterative model. In the next section we will discuss about the foundations of iterative model, designers journey, validation in iterative development and application of model for airborne applications. The paper also discusses the conclusions at the end.

Keywords: Software process, life cycle, process model, iterative development, pipelining.

I INTRODUCTION

Software projects generally make use of a process to enable execution of the various engineering tasks to achieve the goal of delivering a software product that satisfies the customer requirements. The processes so utilized conform to a process model which represents a networked sequence of activities and objects along with strategies for accomplishing the software evolution. A process model generally comments about the various stages to be executed and any other constraints and condition on the execution of stages. The most common model is waterfall model in which different phases of requirement specification, design, coding and testing are carried out in sequence. The waterfall model was first proposed by Royce who suggested that there must be multiple distinct stages in a project execution. Even if the waterfall model proposes a sequential execution of stages, Royce had also pointed out the necessity of feedback from testing to design and from design to early stages of requirements.

Though waterfall model became the most influential process model, it has some predominant limitations(Boehm). The biggest limitation of waterfall model was that it assumes the requirements are stable and known at the beginning of the project. The phenomenon of requirements being not changed unfortunately does not exist in reality in research and development projects. Instead the requirements do change and evolve during project execution. For accommodating the changes in requirement while executing the project using waterfall model, organizations usually define a change management process. Another major limitation is that it follows the approach of software delivery in one shot at the end. And till the end, no working system is delivered. This in fact involves heavy risks as the users do not have any idea of the system till the very end. To overcome these limitations an iterative development model can be utilized. In an iterative development model software is built and delivered to the customer in multiple iterations. Every iteration delivers a working software system which is an increment from the previous delivery. Iterative enhancement(Basili and Turner) and spiral(Boehm)are two very well known process models that supports iterative development. Agile method of programming (Cockburn)and eXtreme Programming(XP)(Beck)also promote iterative development. In XP methodology the key practice is to deliver software in small iterations.

With iterative development the shorter release life cycle reduces much of the risks associated with one shot delivery. Also requirements need not be entirely

specified at the start of the project, as it is the fact in a complex and newly conceived project. The constraints and requirements can evolve over time and can be incorporated as feed back in the system in coming iterations. Incorporating the changes is easy as any new requirements or change requests can be passed on to a next iteration. That is how iterative development is able to handle shortcomings of the waterfall model and is well suited for complex scientific projects despite having some of its own drawback.(for example, it is practically hard to preserve the simplicity and integrity of the architecture and the design)

II FOUNDATIONS OF ITERATIVE DEVELOPMENT

There are basically two successful, accepted and yet apparently opposite practices in software engineering[3]. Do it right the first time - This practice is originated with extreme optimism and self confidence of non failure. In software engineering this practice is clearly reflected in waterfall model. The Fail fast - This practice is originated in pessimistic belief that problems are part of reality. As far as software engineering is concerned, this practice is the base of spiral and other iterative or incremental development models.

Iterative development and water fall approach are generally classified as they have nothing in common. But in reality most of the projects are a mixture of these two opposite concepts. Most frequent complaint or criticism about iterative development techniques may be that it was a poor design in the previous iteration. The whole purpose of the iterative development is to find the risks hidden in the design of the previous iterations, correct it sooner and progress further. Its purpose is to discover pitfalls that otherwise could not have been foreseen.

Sequential development relies on designing the whole system first and then building all parts which at the end must perfectly fit into the perceived system. For this to happen the detailed foreknowledge of the system, various components, their interaction, user perception etc is mandatory, whereas intellectual difficulties are inherent in a complex and evolving system. Thus iterative development becomes the method of choice for building novel high complexity systems. But the size of iteration, the effort and money involved in each iteration, validating the output with requirements and finding nonconformance and finally zero in on to the bugs and corrective actions require lot of insight and expertise. However drawing a line between iterations is always a dilemma. In this context Tom Glib offers a rare piece of guidance - ”the juiciest one next”. Understanding the heuristics helps provide vision and focus which is essential when building complex systems.

III A DESIGNER’S JOURNEY

The essence of a software architecture is contained in the relationships between the different elements present in the system. Hence it is utmost important to build the architectural frame as a first step with only as much functionality as required to verify that the frame is appropriate for the system. Once the elements are integrated it boosts confidence and once understood it provides more insight in to how and where the subsequent pieces has to be tailored. Researchers have found that one of the most prominent risk in building complex system is excessive unrealistic or unstable requirements. The key challenge is to identify the areas of volatile requirements at the earliest and should provide room for resilient to even dramatic changes in requirement or functions. An airborne radar system for example deals with the understanding of the natural environment(the earth, space, water or meteorological conditions), the aeronautical environment(with navigational aids), the radar system and communication with other external systems.

Building the frame first means that the early focus must be on establishing infrastructure that supports necessary interaction between various application objects. The frame should be flexible when needed and stiff wherever necessary-to accommodate corrections before proceeding further. This idea is well explained by Drasko Sotirovski in his paper titled “Heuristics for iterative software development” [1].

The most common iterative development approach comprises of a sequence of iterations with each of the iterations delivering parts of the requirement functionalities. Even though the functionalities are delivered in parts, the total development time is not reduced. If we wish to reduce the total development time, a natural approach is to use parallelism between the different iterations. That is, a next iteration begins before the output produced by the current iteration is released and hence development of a new release happens in parallel with the development of the current release. This model ensures that deliveries are made with a much greater frequency thereby substantially reducing the cycle time for each delivery.

As pipelining is to be employed to achieve parallelism the stages of iteration must be carefully chosen. The stage should be such that its output is the only thing needed for the team performing the task of the next stage with minimal communication. As an

example, consider an iteration stage consisting of requirement specification, coding and testing. The requirement stage is executed by its team of analysts and end with a prioritized list of requirements to be built in this iteration. The requirement document is the main input for the implementation team. This team implements these requirements and hands over to the testing team after performing developer level testing. The tested code is then sent for deployment or integration with other subsystems. The figure1 shows the comparison of waterfall model with iterative and the incorporation of parallel stages in iterative model.

is important to create mission critical pieces as well as more frequently visited functionalities at the earlier stages of iteration itself. Early iterations should focus not only on implementing these requirements but on implementing a frame work resilient to requirement changes. This confirms the property of early functionality which has more time to mature and gain the quality that comes with age.

IV SAMPLE APPLICATION

The iterative model with much of parallelism between stages of iteration has been used for the under mentioned

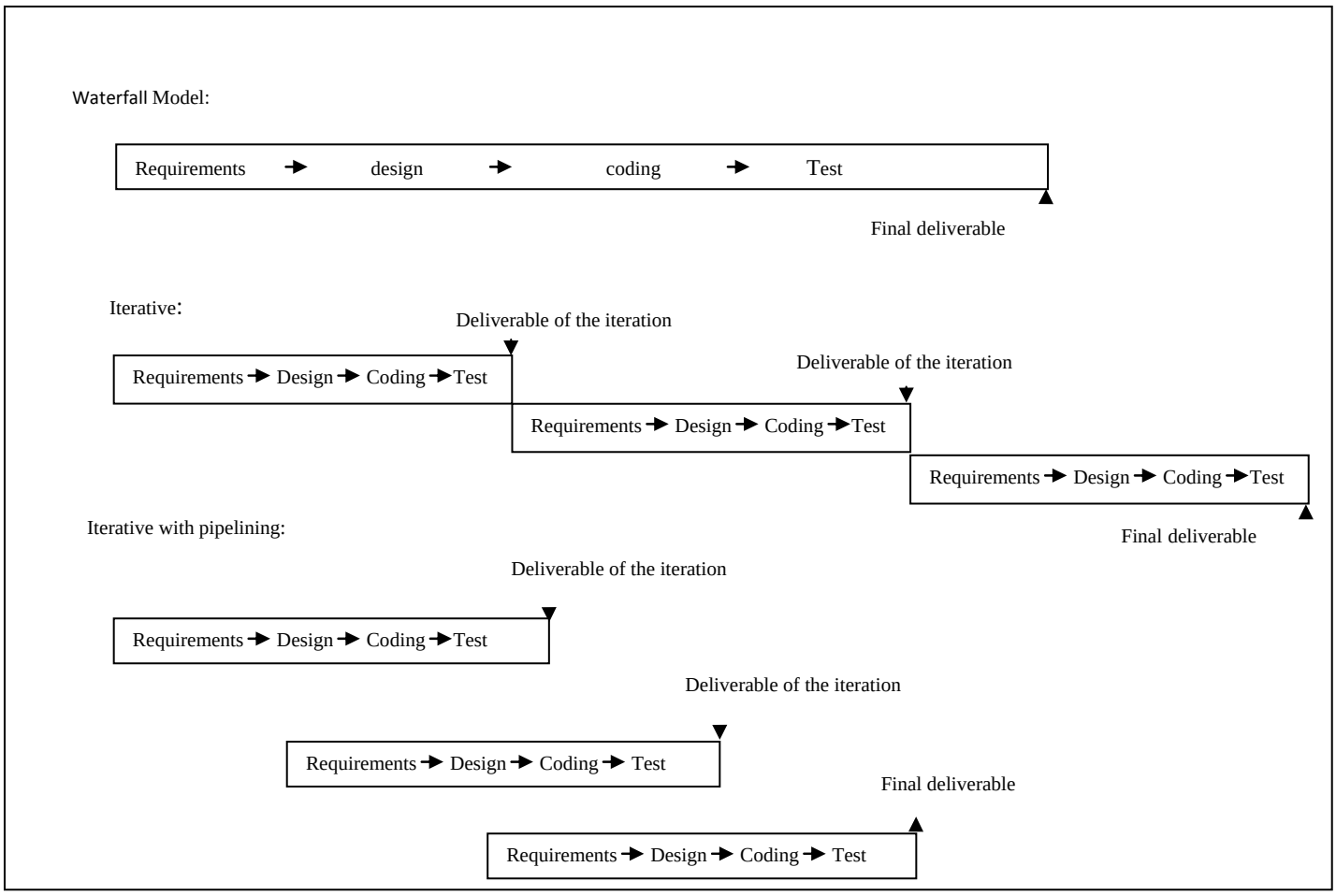

Figure1(courtesy paper on process model by Pankaj Jalote and associates in the journal of Systems and software)[2].

V VALIDATION IN ITERATIVE DEVELOPMENT

The essence and motivation of the iterative development is to reduce risk by validating the proposed design as early as possible. Anything less than integration goes against the iterative development principles because the intention is to discover problems that otherwise could not be foreseen. For these reasons it

application ie, in airborne radar software development which consists of multiple subsystems. As and when the requirements are captured at high level, the subsystem requirements are derived. The first step is to identify the requirements which are independent of each other and are major as far as the customer requirements is concerned. In other words, these requirements should be less likely to be changed and can be tested after its completion so that its test results play a major role in the

system characteristics. For this iteration the derived subsystem requirements are delivered to the implementation team of each subsystem. As on the requirements are handed over the requirement capturing team concentrates on next set of derived requirements, interface requirements, implicit requirements and so on. Mean while in parallel approach, the implementation team completes coding and delivers the piece of code to the testing team. The testing team performs the test cases and verifies against the requirements.

This way multiple stages of iteration can progress independently and in parallel. At this stage the support software for testing also has to be considered as a major requirement. The testing can be performed either by the in-house developed simulator or Commercial Off The Shelf(COTS) simulators according to the suitability of the application. The choice of the simulator should be such that it should be able to generate the required environment for the major functionality which is under test. For example in an airborne radar development application we have developed an environment simulator in parallel with the application development by a different team so that by the time the major functionality is ready for testing the simulator was made available for the test. Once the code to be tested is handed over the next stages of requirements in pipeline are considered and implementation begins for that.

As and when the test results and the review points are available, the changes are made part of the next iteration. As per the severity and dependency of these results, it is also possible to make these part of the second iteration. But since the first iteration is chosen in such a way that it is more independent, pipelining the required changes as part of a third iteration would also serve the purpose with no impact on the planned way of stages of iteration.

As and when each subsystem is ready with primary set of requirements, the subsystems can be integrated and tested. Even users are involved at this stage which helps to find out the ground realities in a complex project with fresh hand experience. This first iteration can be named as a first release which will bring out the risks and practical implications and what is different from the assumptions in the simulation. After this stage, it also may be required to change the simulator software which gives results as that of a practical situation. The radar environment simulator which we used has been undergone such changes after the first integration and trials.

In total, the parallel approach we adopted in iterative development has helped in finding the risks and bugs as early as possible and take corrective actions. The project requires tight configuration management as many teams are working concurrently. The reconciliation procedures need to be solid and applied regularly as it is likely that changes will be made by the team for a stage to the output produced by the previous team. And when this is done, as the previous team is already working on the next iteration there will be a need for reconciliation. This is quite likely to happen between the build and deployment stages as the bugs found during deployment are typically fixed during next iteration.

CONCLUSION

It is a necessity to manage risks in an effective way. Though the features to be built should be decided on priority, the learning curve that is needed in a complex system is to be accounted for. The light weight features chosen in first iteration will enable the team to become familiar with domain and the existing system and hardware. Requirement changes can be handled as per the model-unless urgent they can be pushed to the next available iteration. For bug fixes, unless the bug is critical(in which case it is corrected immediately), the bug report can be logged and scheduled as part of the requirements for the next iteration. The determination of functionalities in first iteration plays an important role and then plan the effort and schedule for delivering the functionality in that iteration. Due to pipelining the turnaround time for each release is reduced substantially without increasing the effort requirement. As discussed above to keep the project manageable, the number of stages in iteration should be a few.

REFERENCES

[1]Heuristics For Iterative Software Development, Satirovski (D), IEEE Software 18,3;2001, May/Jun; 66-73.

[2]The Time Boxing Process Model For Software Development, Pankaj Jalote, Department of Computer science & Engg, IIT Kanpur, India.

[3]Software Engineering, A Practitioner's Approach, Roger S Pressman, 7thEd, 2010.

BIO DATA OF AUTHORS

Prasanthi E R was born in 1980 at Thrissur, Kerala. She graduated in Computer Science and Engineering from Govt. Engineering College, Thrissur in 2001. She is working in the area of Radar Data Processing in Electronics and Radar Development Establishment since April 2004.

Aparna Rathi**,** received the M.Tech degree in Electronics Design Technology from Center of Electronics Design Technology of India, Aurangabad, in 1997. She is a scientist in

Electronics and Radar Development Establishment, Bangalore since 1999. She has worked for development of target tracking and radar data simulation software for different radar applications.

J.P.Vardhani studied MSc in Andhra University, Vishakhapatnam and M.S. in Software Engineering at BITS, Pilani. Joined DRDO in 1990. Currently working with LRDE (DRDO). Area of specialization is Radar Controller and Display. Areas of interest include Real Time Systems, Software Engineering and Artificial Intelligence.

Vivek Krishna was born in 1979 at Ballia, Uttar Pradesh. He completed M. Tech in Electronics Engineering in 2004 J.K Institute of Technology, Allahabad. He worked as a Lecturer at BBS Engineering College, Allahabad and ISDC, Allahabad. He joined Electronics and Radar Development Establishment, Bangalore in 2008 as a scientist and has been working on radar data processing applications.