IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661,p-ISSN: 2278-8727, Volume 18, Issue 3, Ver. I (May-Jun. 2016), PP 74-78
www.iosrjournals.org

A Quick Development Model for Change Oriented Software Process

[1]D. Hema Latha , [2]Prof. P. Premchand
[1]*Asst. Professor, Dept of Computer Science, University College for Women, OU, Koti, Hyderabad, TS, India*
[2]*Professor, Dept of Computer Science and Engineering, UCE, Osmania University, Hyderabad, TS, India*

Abstract: *Changes are common for software process today and hence research is required for change-oriented software engineering. Increase in maintenance costs have become a major concern for developers and users of software systems. Changeability is an important aspect of maintainability, especially in environments where software changes are frequently required.*
A quick or an agile development model is invented for handling changes. In this paper a quick methodology based model for change-oriented software engineering is presented and various model execution environments are also discussed. The key benefit of this quick methodology is used to simplify the change oriented software engineering process.
Keywords: *Change oriented software engineering, software quality, maintenance, change impact, design and software metrics.*

I. Introduction

In typical software development process it is assumed that all the requirements are complete and can be implemented directly in order to develop the application, but this is not the case for most of the projects today. In this competitive world changes are frequent to any software product or module which is under development, due to the market competitions. Priority of requirements changes frequently and only specific development is done which is urgently required and then later on changes and improvements or change oriented models [1] comes into the picture for the remaining developed modules.

So requirement engineering is done in parallel by using tools and techniques to software development and requirement changes often happen to survive in the competitive market.

Whenever there is a request for a requirement, it needs lot of effort in terms of time and cost for analysis and implementation. Theoretically change requirements takes less time than typical development requirements but practically it takes almost the same or even more time as development for complex change requirements. Quick or Agile development’s refactoring and testing practices accommodate software evolution. [3].

II. Need And Importance Of Research Problem

Change requests of any type simple or complex needs a complete software development lifecycle, because after analyzing the requirement it is implemented and integrated with the existing code and then implemented requirement is verified against the test cases and also verified against the functionality required.

Once implementation is done and verified, lot of refactoring is required. Hence this refactoring or restructuring often forces the application to undergo a complete development cycle, including unit, acceptance, and regression testing, followed by subsequent redeployment.

In a large IT or engineering production system, this can be time consuming and error prone. So to overcome this problem, work in this research area is of great importance.

III. Objective

This quick model is designed for changes, without refactoring and rebuilding. Its objective is to design programs that are receptive to change. Ideally, quick programming allows changes be applied in a simple, localized way to avoid or substantially reduce major refactoring, retesting, and system builds.

Some of the most commonly used agile techniques are discussed in this section. There are several parameters associated with the choice of these agile techniques, some of them are team size, iteration length and support for distributed environment. These parameters are also discussed for the most commonly used agile techniques [4].

IV. Agile Methods

Few agile methodologies [5] are discussed below:

A. Extreme Programming (XP)

The important principles of Extreme Programming (XP) are communication, simplicity, feedback, courage, and quality work. Extreme programming is a good quick or agile method and can be implemented when the team size is generally small i.e. from 2 to 10. Iteration length is generally short around 2 weeks. XP is not suitable for distributed teams. Example of Extreme Programming (XP) is given in figure. (1).

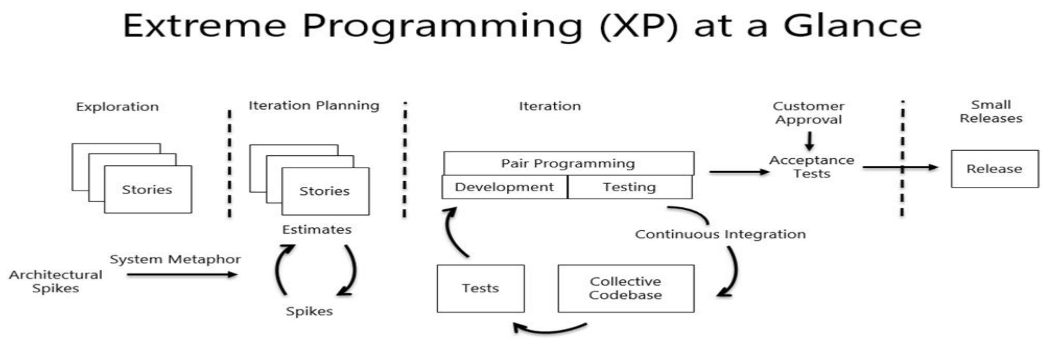

Fig. (1). Extreme Programming (XP)

B. Scrum

Scrum, is one of the most widely used Agile Method, along with XP. Scrum projects are split into iterations (sprints). Development team is divided up to 7 persons team. Iteration length varies from 1 week to 6 weeks. Project may consist of multiple teams that could be distributed. An example for Scrum process is shown in fig. (2) and Scrum framework is shown in fig. (3).

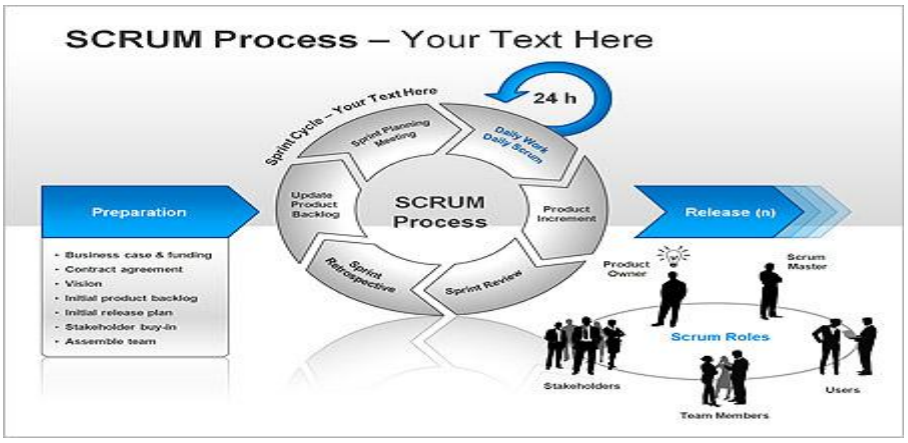

Figure. (2). Scrum Process

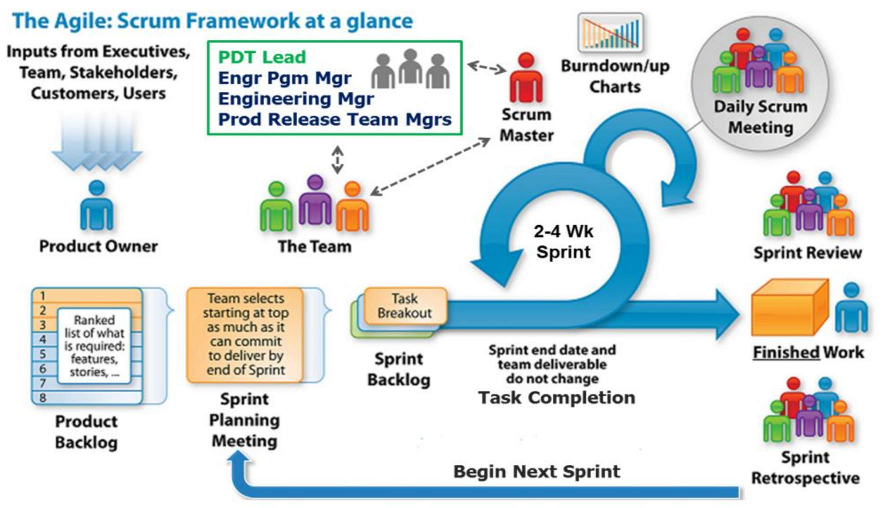

Figure. (3). Scrum framework

C. Crystal family of Methodology

Crystal methods are dependent on written communication or documents communication to achieve maximum extent and can be reduced to a verbal communication for faster development. All Crystal methods begin with a basic set of roles, work products, techniques, and notations. There is no limit on team size in crystal methods. Iteration lengths are generally for [4 months and more. It is build to support distributed team.

D. Feature Driven Development

Feature Driven Development is the feature-oriented development approach that came to be known as FDD. FDD process is the most simplistic development process this works in three steps - i. Develop a development model ii. Build a feature list and iii. Plan by feature. The team size varies as per the features complexity. Iterations lengths are up to 2 weeks but it do not have support for distributed systems.

V. Proposed Model

When a new change request comes, it should be analyzed and the change request type should be identified. New change can be requested at any phase during development i.e. at the beginning of the project or in the middle of development or during testing or once the product is

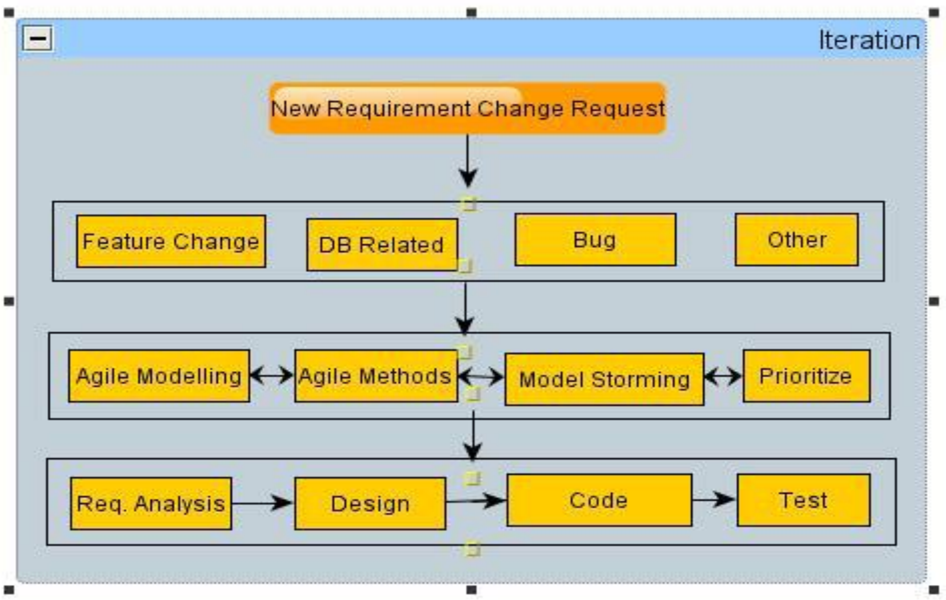

Fig.4. One iteration for new requirement in agile SDLC model.

Released and due to market standards and any other associated parameter change, changes in the product. So the model should be flexible to support the new change request at any stage of development. Figure 4 depicts the one iteration of proposed model.

When a new change requirement will come, first it is categorized, means it is filtered in terms of functional requirements, the new change request could be a change in the feature, it could be database related change or it could be a bug or defect in the system or it could be change due to any other reason i.e. change in specifications etc. Once the change request is categorized, it will be placed into an appropriate agile model and technique for this. Next model storming is done so that in next time if same type of change request comes it can be handled faster. The change request is prioritized in terms of execution. In the last phase this change request is sent to the SDLC for the execution. Test driven development (TDD) is also integrated with the agile methodology for faster and accurate development of requirements. It also supports the change request implementations more accurately in a faster way. The TDD for a change request is explained in figure 5. In TDD all the test cases are written in advance and then implementation is done according to the written test cases. For every new change requirement all the test cases are written before hand (test cases are modified if they exist already in the system) and then they are verified against the functional requirements. Once the test cases verification is done, they are implemented and the implemented code is re-factored as per the coding specifications i.e. implemented code is locally shifted to merge to form a package etc.

The project is developed in phase wise, hence the incremental software models are very popular because of its simplicity of executing the requirements in step wise or phase wise. And most of the designers prefer this model because the execution is in increments wise. Since the requirements are not clear to the development team at the beginning and generally requirements change time to time and may get clear to the team at middle of the development, so incremental model can be is used towards the successful execution of the project.

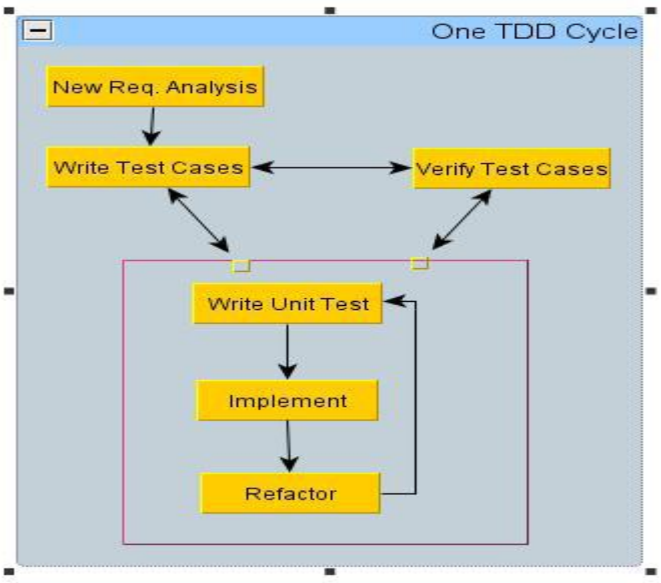

Fig. 5. Test driven development cycle for a new requirement.

The incremental model can be integrated with the quick or agile methodology to get benefit from both the techniques. Figure 6 represents the quick incremental model for the change request handling. Figure 6 represents an N iteration quick incremental diagram. Iteration 0 is the basic model of beginning stage, where actually the model is evaluated against the execution environment and suitable agile development methodology is adopted as per the specifications. Iteration 0 requires lot of brain storming and analytical work since it is going to be repeated by all the consecutive iterations. Iteration 0 is explained in figure 7 and is discussed in next paragraph. The rest iterations are a complete agile SDLC lifecycle which is depicted in figure 4.

Figure 7 represents the iteration 0 for the agile incremental model. This is the initial phase where the model is analyzed properly and made ready so that it can be executed in increments. It requires lot of analytics and study of various methods which can be used in long run of the model. The iteration 0 mainly consist of initially modeling of the given change requirements and a rough model is decided and evaluated against the number of affecting parameters for the system which is done under model storming, here the suitable agile techniques are also decided for the model. As per the requirements extreme programming (XP), crystal methods or test driven development (TDD) [6] is chosen, the most effective agile development technique is TDD. Finally level 0 is also going to deal with the architecture and design modeling of the system which will be followed by N numbers of iterations.

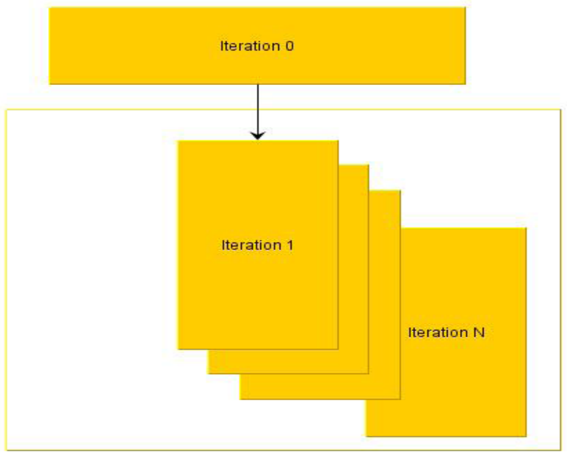

Fig. 6. An incremental agile SDLC lifecycle.

The initial modeling can be one of Usage model, Domain model or User interface model as per the requirements. Usage modeling deals with the actual usage of the implemented requirement, Domain modeling deals with analyzing the domain and parameters related to the domain related to the requirements and User interface modeling deals with the end GUI interfaces etc for the requirements. The model storming could be Analysis Model Storming, Design Model Storming or Adopting Model Storming depending on the need of requirements. Analysis Model Storming deals with the detailed analysis of the change requirements, Design Model Storming deals with the design aspects of the project, because of which new change requirements can be affected and Adoptive Model Storming deals with the execution environment aspects of the model where the requirement changes model will be executed [7].

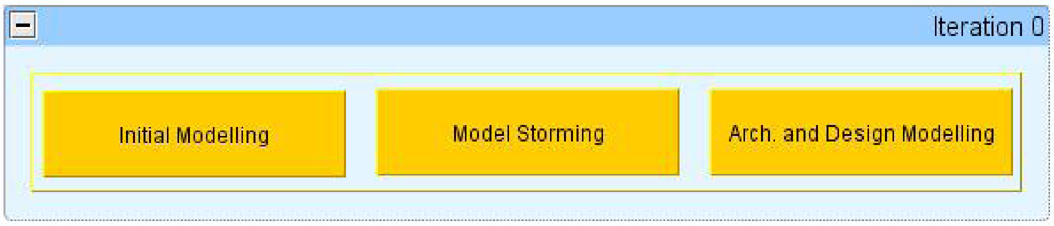

Fig. 7. Initial iteration for incremental agile SDLC lifecycle.

The prioritize model for the newly arrived change request [8] is explained in figure 8. The priority of a new change request can be decided by considering the various relevant parameters. The most common parameters are current open tasks list, open bug list, customer inputs regarding the characteristics of newly change requirement, development team's input for the new change request in terms of whether this new change request is going to affect the existing design etc., and dependencies on other open issues. Based on these parameters the priority of the new change request is calculated and high priority task list is generated.

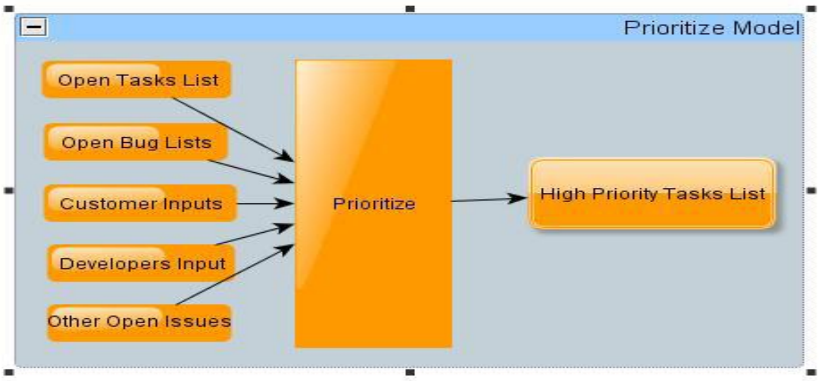

Fig. 8. Change Requirements Priority Model.

VI. Conclusion

In this paper the advantage of quick or agile development is adopted to simplify the change-oriented software engineering. A quick or an agile methodology based change oriented software engineering model is proposed. In this paper other various model execution environments are also discussed related to the proposed model.

References

[1] Peter Ebraert, Jorge Vallejos, Pascal Costanza, Ellen Van Paesschen, Theo D'Hondt,, "Change-Oriented Software Engineering", ACM International Conference Proceeding Series; Vol. 286, Pages 3-24, 2007.
[2] T. M. Pigoski. Practical Software Maintenance. John Wiley & Sons, New York, 384 pages, 1997.
[3] Romain Robbes and Michele Lanza, "Change-based Approach to Software Evolution", Electronic Notes in Theoretical Computer Science (ENTCS) archive, Vol. 166 , Pages 93-109, 2007.
[4] Dave Thomas, "Agile Programming: Design to Accommodate Change", IEEE, Vol. 22, No. 3, May/June 2005.
[5] Laurie Williams, "A Survey of Agile Development Methodologies", pp 209-227, 2007.
[6] Scott W. Ambler, "Agile Model Driven Development (AMDD)", XOOTIC MAGAZINE, February 2007.
[7] Jeffrey A. Livermore, "Factors that Significantly Impact the Implementation of an Agile Software Development Methodology", ACADEMY PUBLISHER, 2008.
[8] Par Emanuelson, "An agile modeling environment for partial models", agile alliance publications, pp 223-224, 2002. © 2009 ACADEMY PUBLISHER