

An Agile Methodology Based Model for Change-Oriented Software Engineering

Naresh Kumar Nagwani, Pradeep Singh
 Department of Computer Sc. & Engg.
 National Institute of Technology, Raipur
nknagwani.cs@nitrr.ac.in, psingh.cs@nitrr.ac.in

Abstract- Changes are common to software development models today and hence change-oriented software engineering is in the picture, in the area of research. Agile development is invented for handling change. In this paper an agile methodology based model for change-oriented software engineering is presented and various model execution environments are also discussed. The key benefit of agile methodology is used to simplify the change-oriented software engineering process.

I. INTRODUCTION AND RELATED WORK

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 modern competitive era 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 comes into the picture for the rest developed modules. So requirement engineering is done in parallel to software development and requirement changes often happen to survive in the competitive market.

Whenever a new requirement comes into the picture it takes 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. Since changes of any type whether 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 related work is required for making sure that the implemented code is written in standard format and integrated with the system as per the development standards. Refactoring is also an important type of change requirement which sticks the development policies with the developed code and which

comes into the picture once the development task in bulk is over. Refactoring improves code, usually increasing the function while reducing code bulk. However, such 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.

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

In this section work related to change oriented software engineering is discussed in brief, and in the subsequent sections various most common agile techniques, various model execution environment and proposed model is elaborated.

A change model for Change-Oriented Software Engineering (COSE) is proposed in [1]. Based on an evolution scenario, a lack of support in current Interactive Development Environments (IDEs) is identified to apply COSE. A set of extensions to an existing model of first-class changes and describe the desired behavior of change-oriented IDEs to support COSE is introduced.

Model development information as change operations is proposed that is retrieve directly from the programming environment the developer is using, while developer is effecting changes to the system. This accurate and incremental information opens new ways for both developers and researchers to explore and evolve complex systems. [2]

Systems in use today (including CVS and Subversion) are indeed losing a lot of information about the system they version. Main, orthogonal, reasons are identified: most versioning systems are (1) file-based, and (2) snapshot-based. The proposed approach by Thomas is based on two concepts:

(1) an IDE integration to record as much information as possible and to allow easy access to our tools, and (2) a model based on first-class change operations to better match the incremental process of developing software. Applications are expected to evolve over time as their requirements change, and agile development's refactoring and testing practices accommodate software evolution. [3]

The principle of separating policy (what) from mechanism (how) is a best practice in design. A policy-driven system uses a set of rules to describe a specific policy and an associated specialized policy "language" runtime that provides the execution mechanism. A number of the factors that impact the implementation of an agile development methodology are completely under the control of management. Organizations that are considering implementing an agile methodology are able to manipulate some of these factors to increase the opportunities for success of their methodology. [6]

II. THE AGILE TECHNIQUES

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 these most commonly used agile techniques here:

A. Extreme Programming (XP)

Five key principles of eXtreme Programming (XP) are communication, simplicity, feedback, courage, and quality work. Extreme programming is a good agile methodology 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.

B. Scrum

Scrum, along with XP, is one of the more widely used Agile Methods. 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.

C. Crystal family of Methodology

Crystal methods are based on the principle that how to achieve a maximum extent by which a written communication or documents communication can be reduced to a verbal communication for faster development. All Crystal methods begin with a core set of roles, work products, techniques, and notations. There is no limit on team size in crystal methods.

Iteration lengths are large generally 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.

III. THE MODELLING ENVIRONMENT

For any model, environment plays a critical role. Environmental parameters are going to affect the execution of a model, some of the types of model environments are mentioned in this section, which takes cares for execution of any model. The modeling environment for agile development is given in [7] by Par Emanuelson. The agile development environment is divided in three main categories – i. Direct Execution ii. Partial Models and iii. Incremental Execution. These environments are explained in the next paragraphs.

Direct execution refers to the execution of model in the environment without considering the factors like interrupting events etc. Hence the user will free of thinking the model and expected outcome of the model. Model itself contains all the required information for the execution like action language code for methods etc and all the required code is generated in advance.

Before a model will be executed all of the required parameters and dependencies should be provided in order to successful execution of the model. Some times there could be the situations when the all the needed parameters etc are not supplied during execution time. An example of this situation is dynamic type checking mechanism where the data type is decided at run time and at compile time the system or model do not have any information regarding the data type at runtime it is decided. For these types of requirements there is Partial Models environment systems.

In direct execution the model should be ready for execution, so that it can be directly be executed in the environment. Which is not possible in all of the cases instead user might be interested in executing the model step by steps by slightly modifying the relevant parameters, here the second technique Incremental Execution Models execution comes into the picture for achieving this behavior.

For simple SDLC life cycle the direct execution environment methodology is used, whereas for incremental models, incremental execution techniques are used. The most common choice is the partial model execution and incremental model execution techniques since agile is invented for change, where the needful information is not available at beginning, hence partial model or incremental model execution is obviously a useful choice for the model.

IV. PROPOSED MODEL

New change request can come at any point of development i.e. at the start of any project or at the middle of development or during the bug fixing for a project or once the project is released and due to market standards and any other associated parameter change, changes in the product. So the model should be enough flexible to support the new change request at any moment of time.

Figure 1 depicts the one iteration of proposed model. When a new change requirement will come into the system first of all it is categorized. It means it is filtered in terms of functional requirements, the new change request could be feature change request, it could be database change related request or it could be in terms of 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 newly change request is classified it will enter into the second phase where it is fitted to a suitable agile model and technique for this. Here model storming is also done so that in next time same type of change request can be handled faster. The priority of the change request is also decided in terms of execution. In the last phase this change request is sent to the SDLC cycle for the execution.

Test driven development (TDD) is the modern art of development which is also integrated with the agile methodology for faster and accurate development of requirements. It also supports the change request implementations more accurately in faster manners. The TDD for a change request is explained in figure 2. 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 first of all the test cases are written (test cases are modified in case they already exist in the system) and then they are verified against the functional requirements. One test cases are verified they are implemented in simple manners and the implemented code is refactored as per the coding specifications i.e. implemented code is locally shifted to clubbed together in some package etc.

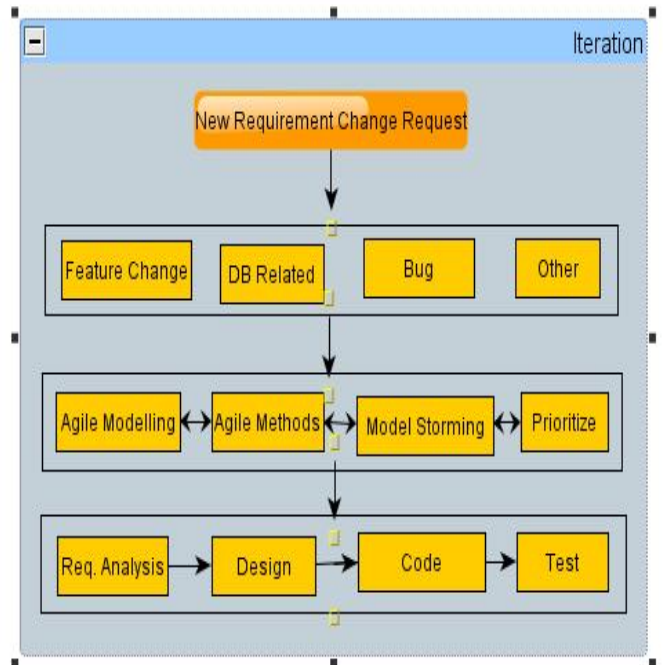


Fig.1. One iteration for new requirement in agile SDLC model.

Any project is developed in stages hence the incremental software models are very popular because of its simplicity of executing the requirements in stages. And most of the designers prefer this model because of its properties of executing in steps. Since most of time all the requirements are not clear to the development team and requirements generally changes at time to time and are clear to the team at middle of the development, so incremental model support the development team to provide this type of environment where the efforts can be utilized towards the successful execution of the project.

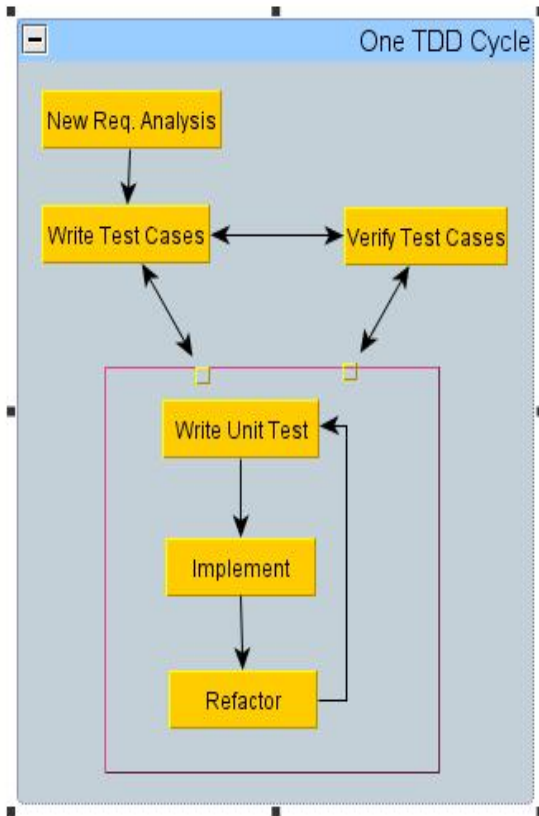


Fig. 2. Test driven development cycle for a new requirement.

The incremental model can be integrated with the agile methodology to get benefit of both of the techniques. Figure 3 represents the agile incremental model for the change request handling. Figure 3 represents an N iteration agile incremental diagram block diagram. Iteration 0 is the basic model preparation stage, where actually the model is evaluated against the execution environment and suitable agile development methodology is adopted as per the task list in hand. 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 4 and is discussed in next paragraph. The rest iterations are a complete agile SDLC lifecycle which is depicted in figure 1.

Figure 4 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 ready to execute in stages. 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) 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.

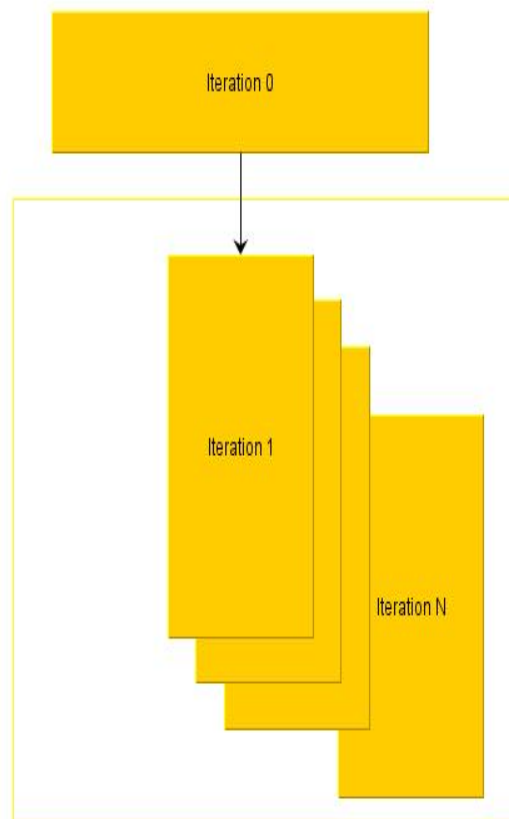


Fig. 3. An incremental agile SDLC lifecycle.

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.



Fig. 4. Initial iteration for incremental agile SDLC lifecycle.

The prioritize model for the newly arrived change request is explained in figure 5. 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.

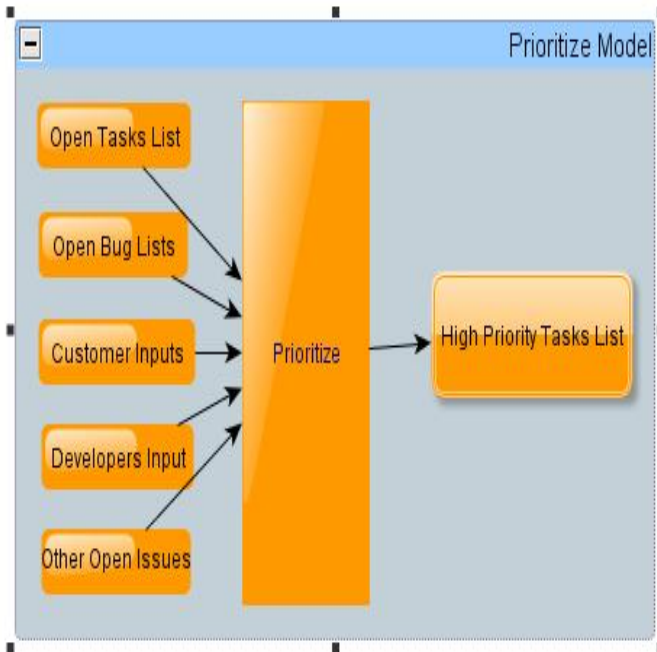


Fig. 5. Change Requirements Priority Model.

V. CONCLUSION

In this paper benefit of agile development is adopted to simplify the change-oriented software engineering. An agile methodology based change oriented software engineering model is proposed. Various model execution environments are also discussed for 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] 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.
 [3] Dave Thomas, “Agile Programming: Design to Accommodate Change”, IEEE, Vol. 22, No. 3, May/June 2005.
 [4] Laurie Williams, “A Survey of Agile Development Methodologies”, pp 209-227, 2007.
 [5] Scott W. Ambler, “Agile Model Driven Development (AMDD)”, XOOTIC MAGAZINE, February 2007.
 [6] Jeffrey A. Livermore, “Factors that Significantly Impact the Implementation of an Agile Software Development Methodology”, ACADEMY PUBLISHER, 2008.
 [7] Par Emanuelson, “An agile modeling environment for partial models”, agile alliance publications, pp 223-224, 2002.