An Agile Development Model with Enhanced Security for Change Oriented Software Process

[a]D. Hema Latha , [b]D. Rama Krishna Reddy, [c]Azmath Mubeen

[a]Asst. Professor, Dept of Computer Science, University College for Women, OU, Koti, Hyderabad, India,
[b]Asst. Professor in Computer Science, Dept of Mathematics, UCS, Osmania University, Hyderabad, TS, India,
[c]Asst. Professor, Dept of Computer Science, University College for Women, OU, Koti, Hyderabad, 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.
In most organizations, now-a-days, software is not secured, contains security susceptibilities that can be exploited by people with malicious intent to cause financial and physical damage. For this reason most research efforts have been involved in general development and maintenance processes, which have led to the implementation of few models. One such model used for software maintenance is maintenance model with security measures.
A quick or an agile development model with security is invented for handling changes and also for providing security. In this paper a quick development model with security for change-oriented software engineering is presented. The key benefit of this quick methodology with security is to simplify and handle the change oriented software engineering process with ease and efficiency.
The study in this paper also identifies the software design issues that need to be addressed during the maintenance stage in order to modify the general maintenance model into quick and security model for software design and maintenance. The proposed model aspires to avoid design susceptibilities by taking security features into consideration. In order to study the suitability of such a model, studies will be conducted with software industry experts, and the results will be analyzed. This quick development model with security model can be used to train software analysts, designers or software developers as how to make changes quickly in maintenance phase and to provide security and to maintain secure software designs, while reducing susceptibility.

KEYWORDS: Change oriented software engineering, software quality, maintenance, change impact, design and software metrics, Secure Software Design, Task Oriented Maintenance Model, and Design Susceptibility, Agility.

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 developed or 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, design 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. Problem Background

Maintenance is considered to be the continuous process of making the system in proper working order till the retirement of the system. With time the system will deteriorate; hence concept of maintenance along with security is necessary. Therefore, the main goal of maintenance is to keep the subject's functionality in tact with that which had been defined and registered at the time of release [1].
Software maintenance is defined in phases as a solution for fixing a problem or bug. The definition is similar to the software maintenance definition proposed by ISO/IEC 14764 (1999). However, the IEEE (1994) has also proposed the following definition [2]: "Software maintenance is the process of modifying a software system or component after delivery to correct faults, improve performances or other attributes, or adapt to a changed environment."Software maintenance is defined as a post implementation activity that is started when a system is released to the customer. This view is well-summarized by the classical waterfall model of the software life cycle, which is shown in Figure 1.

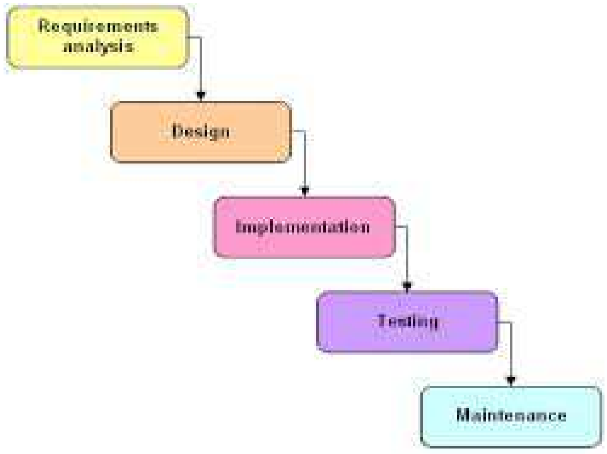

Figure.1. Software development life cycle model

III. 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.

IV. Objective

This quick model with security is designed for changes, without refactoring and rebuilding and also provides security. 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. Once the model is implemented, next is to provide security for the design in order to implement the changes.

V. Building Secure Software

According to (Smith 2009), design weaknesses can be reduced by identifying and capturing security issues through threat modeling and security requirement analysis in the early stages of design architecture or at the time of redesign in maintenance, when requests for changes. Generally, design weakness is a flaw in the software system architecture, the specifications, or low-level or high-level design, and is caused by a security gap or essential oversight during the design stage. This kind of defects or flaws often occurs because of incorrect assumptions made during the run time environment and increases the risk that the system will enumerate during deployment.

Few security issues not related to syntax or code, such as business logic flaws, cannot be discovered in code and necessary to be identified by performing threat models and abuse case modeling during the design stage of the Software Development Life Cycle. Threat modeling is an analysis technique that helps to identify and reduce design weaknesses in a product (Smith 2009).

The software life cycle can be categorized into two parts: a) the initial development of software, and b) the maintenance and operation of software [3]. Software maintenance is an ongoing activity that includes all the work performed on a software system after it has been made operational, till the expiry of that system. The scope of this definition covers the fixing or correction of errors, the enhancement, deletion, and addition of capabilities, adaptation to changes in data requirements and operation environments, and the improvement of performance, usability, or any other quality attribute. Security-focused software maintenance has yet to receive an appropriate level of concentration from management. Proper planning would make the implementation of a structured maintenance process more efficient and effective. [4] Discuss the fact that legacy software is not easy to handle when performing maintenance for security issues. Legacy software has more complex internal structure, often suffers from the implementation of bad coding practices, and has weak documentation. In order to overcome these issues, secure software design is taken into consideration in order to provide a smooth transition from high level design requirements and improve security concerns in software systems.

VI. Proposed Model

A. One iteration cycle for new requirement in agile SDLC model.

When a new change request comes, it should be analyzed and the type of the change request 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 released and

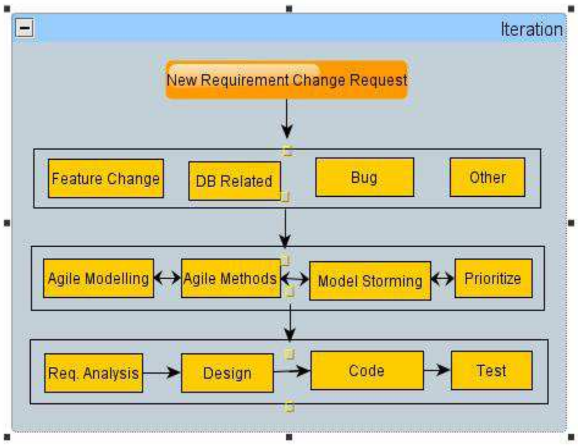

Fig.2. One iteration for new requirement in agile SDLC model.

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, with security concerns. Figure 2 depicts the one iteration of proposed model.

B. Test driven development cycle for a new requirement

When a new change requirement comes, first it is categorized, means it is filtered in terms of functional requirements, the new change request could be a change in the feature, database related change, 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. Next model storming is done so that next time if same type of change is requested c 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 [5] for faster and accurate development of requirements. The TDD for a change request is explained in figure 3. In TDD all the test cases are written in advance and then implemented 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.

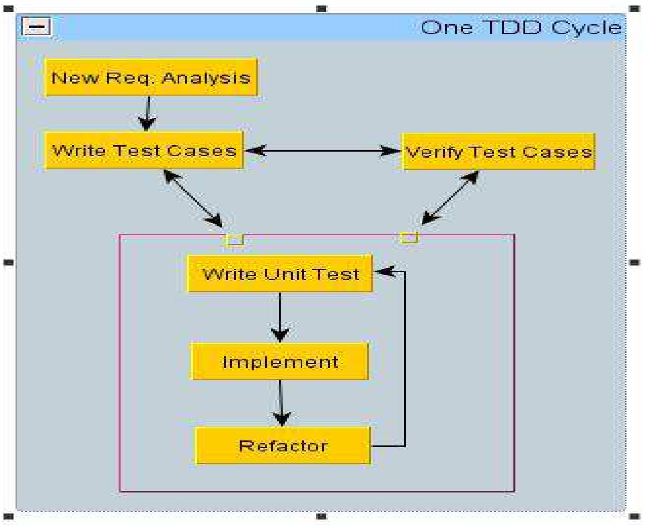

Fig. 3. Test driven development cycle for a new requirement.

The incremental model can be integrated with the quick methodology to get benefit from both the techniques. Figure 4 represents the quick incremental model for the change request handling. Figure 4 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 5 and is discussed in next paragraph. The rest iterations are a complete agile SDLC lifecycle which is depicted in figure 2.
Figure 5 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. 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.

C. An incremental agile SDLC lifecycle

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.

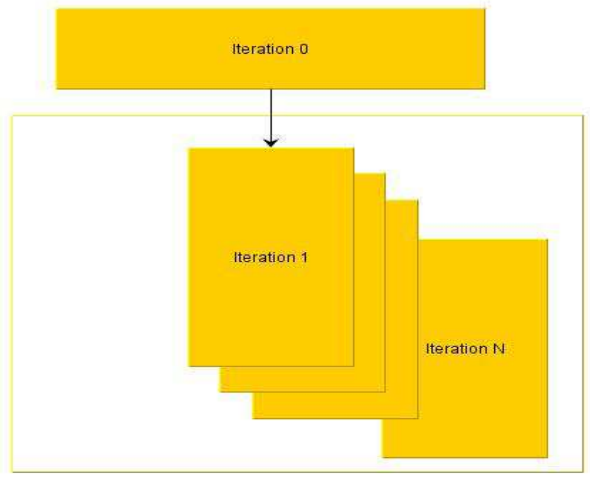

Fig. 4. An incremental agile SDLC life cycle.

D. Initial iteration for incremental agile SDLC life cycle

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].

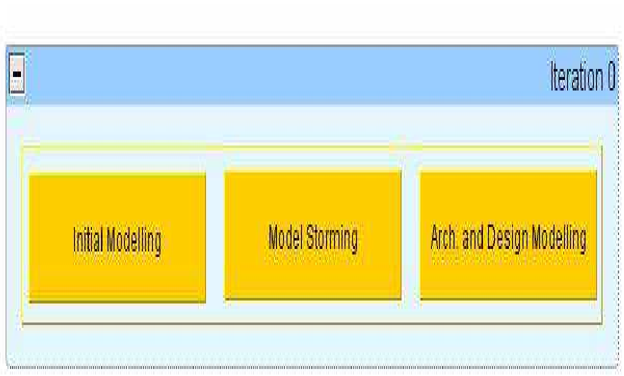

Fig. 5. Initial iteration for incremental agile SDLC life cycle.

E. Change requirements priority Model with security Analysis and measures

The prioritize model for the newly arrived change request [8] is explained in figure 6. 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. Next analysis and measures of security attacks can be assessed.

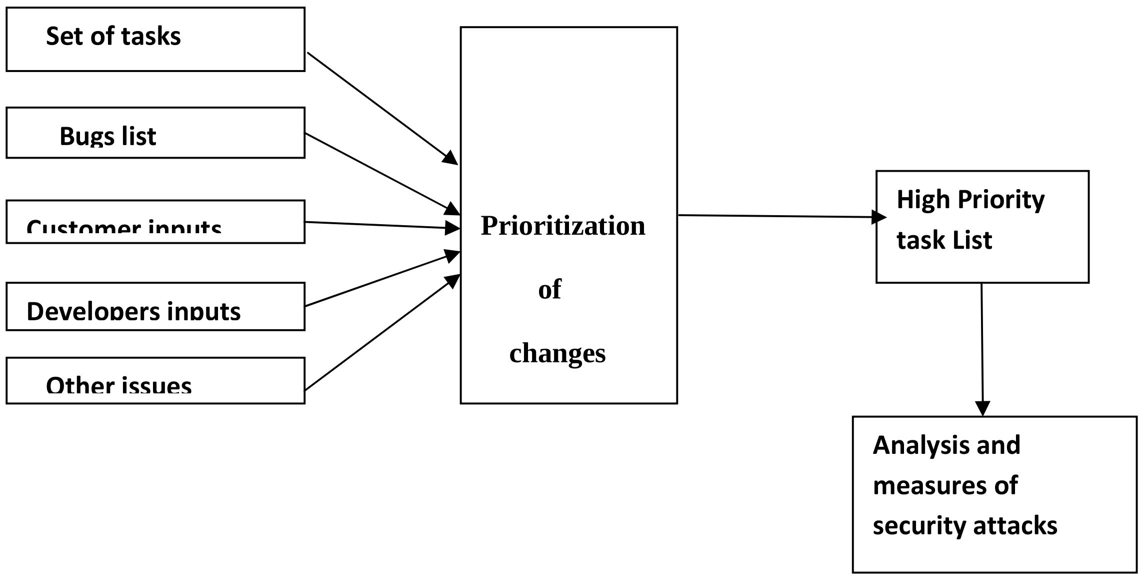

Fig. 6. Change Requirements Priority Model with security analysis and measures

F. Enhanced Security model for software Maintenance

Once the change requests are prioritized, now the change requests requirements and design should be processed in a secured way. According to [9], the objective of secure software development is to analyze, design, implement, configure, and sustain software systems in which security is an important attribute from the starting of the system's life cycle (requirements definition) to its end (retirement). One approach to acquire secure software is to strictly follow to the secure development, deployment, and sustainment to principles and practices. Most serious software vulnerabilities, and the corresponding threats that designers and developers do not pay much attention is to, improper input validation or lack of input validation.

G. Lack of Input Validation

The system should be tested before delivering to the customer, if the input software system is not tested prior to delivery, it can allow an intruder or attacker to provide incorrect information or even inject wrong or illegal code that may be executed by the software.

In the maintenance process when change requests come, after prioritization process, new requirements are added and if any missing requirements there in the design, can be traced from the requirements by traceability process, as the design maintenance process is linked with the requirement process in that functional requirements that were focused on can be connected to the design for secure software. Figure 7 describes the secure software design maintenance process. This process includes security requirements [10], which are typically derived from functional requirements, based on the design. In fact, moreover the secure software design process depends on all the security requirement activities that have occurred, the secure design principles, and guidelines. The results of both processes are equal to the secure design process. If the design system's predicted behavior comes adequately close to its requirements, the designers can proceed to a more detailed level of design or implementation [11]. Analysis of the system design reveals some vulnerability that provides with the reasons why developers have missed few security requirements. Then, the security requirements process is checked again to cover the missed design. The models or the processes related to design can be re-evaluated until the design is secured.

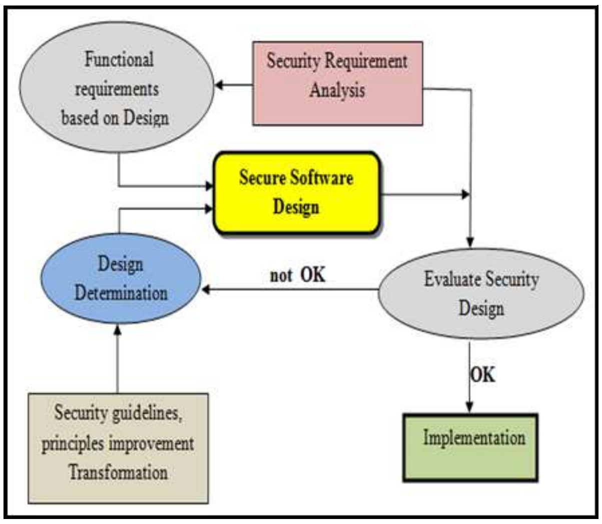
Functional requirements based on Design
Security Requirement Analysis
Secure Software Design
Design Determination
not OK
Evaluate Security Design
OK
Security guidelines, principles improvement Transformation
Implementation

Fig. 7: Enhanced Secure Software Maintenance Process

VI. Conclusions

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 with security is proposed. Here, the proposed model conveys that prioritizing the change requests and by applying good security features the database and open source software systems can be designed easily and therefore the changes can be implemented efficiently and effectively. The proposed model is beneficial for the design of secure software in maintenance phase, especially in web application systems. The proposed model is helpful for databases and open source software systems, also be implemented in any other software industry in the world. Software vulnerabilities can be caused by flaws in software's requirement specifications, design, or source code or with change requests.

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] Dupuis, R.,Software Engineering Body of Knowledge.", 2004.
[9] Peine, H., Rules of thumb for developing secure software: Analyzing and consolidating two proposed sets of rules. Availability, Reliability and Security, 2008. ARES 08, Third International Conference on, IEEE, 2008
[10] Sherman, S. and I. Hadar., Identifying the need for a sustainable architecture maintenance process. Cooperative and Human Aspects of Software Engineering (CHASE), 2012 5th International Workshop on, IEEE, 2012.
[11] Grubb, P. and A. A. Takang, Software maintenance: concepts and practice, World Scientific Publishing Company Incorporated, 2003.