Buletin Teknik Elektro dan Informatika
(Bulletin of Electrical Engineering and Informatics)
Vol.1, No.1, March 2012, pp. 23~28
ISSN: 2089-3191

Designing a Secure Object Oriented Software Using Software Security Life Cycle

Mohammad Obaidullah Bokhari[1], Mahtab Alam[2]
[1]Chairman, Department of Computer Science, Aligarh Muslim University, Aligarh
Uttar Pradesh, India
[2]Research Scholar, Mewar University, Chittorgarh, Rajasthan, India, Ph: 09411653979
e-mail: mubokhari@rediffmail.com[1], alam_mahtab@rediffmail.com[2]

Abstract

The security of object oriented software is well managed by software metrics because it guarantees accurate, reliable, faster and more efficient ways with proven techniques and standard notations. The Software Security Life Cycle (SSLC) was developed and provided a basis to help software security planning. In this paper, software metrics, which have been proposed for Software Security, are used and applied to programming language. In software industries, the cost of Security of large-scale software has put emphasis on the need to manage in the earlier phases, statistically estimate the security of large software system and to identify error prone modules.

Keywords: *Software Security Planning, Software Metrics, Object Oriented Analysis, Object Oriented Design, Software Security Life Cycle.*

1. Introduction

Software security is a major activity in the software industry. It has been reported that cost and effort spent on software security is very high, approximately between 65% to 70% of total software development and support efforts [1]. Software reengineering, recently, have been advocated as a means of reducing security costs [9]. Software security is the degree to which it can be understood, corrected, adapted and/or enhanced existing software. Software security is the costlier activity to all other engineering discipline because of its complexities. Software system has been a dominant influence of successful business, as the growth of scales and contents of the software system [5]. Most of the software systems have become too complex to developed by individual efforts. The development of software usually involves teamwork and needs good communication. However, a lack of validated widely accepted and adopted tools for planning, estimating, and performing security also constitute to the security problem [2].

Object oriented design and programme are the dominant development paradigm for software systems today [3]. The object oriented model closely represents the problem domain, which makes it easier to produce and understand design. It is also believed that object oriented design will encourage more re-use, i.e., new application can use existing modules more efficiently and effectively, thereby, reducing development cost and time [6]. Classes and methods are the basic construct for object oriented technology. The amount functions provided by an object oriented software can be estimated based on the number of identified classes and methods or its variants. Therefore, it is natural that the basic object oriented metrics are related to classes and methods and the size or function points of the classes and methods [4]. The object oriented system have promoted for ease of design, coding and use implying that security and testing are easier and cheaper than other traditional engineering. Object oriented paradigm will significantly increase reusability, extendibility, interoperability and reliability provided which the systems are maintained adequately.

Over the years, researchers have been working on the probable of Software Security to reduce cost. New technologies, such as structured design case tools and object oriented programming, have been adapted as the latest “Silver bullets”, only to prove inert later in practice [10]. The most important factor of the high cost of software security is the lack of close

Received October 6, 2011; Revised November 12, 2011; Accepted January 6, 2012

and effective management of security process. Figure 1 depicts the stages of Software Security Life Cycle (SSLC). Recently, the most of the researchers have shifted from a project-oriented view to a product life cycle oriented view of Software Security and trying to understand the characteristics of security demands. Three different phases were reported in SMLC. Repair rises in first stage, small enhancement in the second stage, and major enhancement climbs in the third stage.

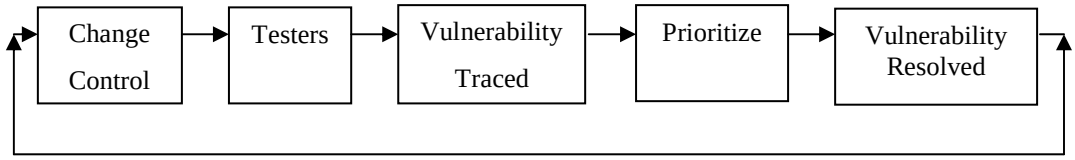

Figure 1. Software Security Life Cycle

2. The SSLC Model and Security Management Framework.

The Software Security Life Cycle is confined to project level (Figure 2) has mainly four stages. The characteristics and function of the stages are as under:

- **Introduction Stage**: The users, in this stage, are those who devoted S/W. During this stage the major concerns are how to make the software works. The security mechanism is concern in this stage.
- **Growth Stage**: In this stage, system usage is up and the number of system attacks reported is important. As the software enters this stage, the monitoring of system attacks and users problem reports is an important concern. In order to mange the increasing security demand, it is critical to allocate more programmers copying with the increasing countermeasure requests to prevent accumulating dissatisfaction. The Security tolerance is the main concern in this stage.
- **Maturity Stage**: After deploying the system, the users experience the growth of enhancement requests in the maturity phase. At this stage, the focus is enhancement of projects. Security experts are expanding the software functions and trying to prolong the life of the software.
- **Decline Stage**: At this stage, the software is easy to vulnerate, and attackers can easily exploit it. The users require software renovation. In this phase, production team must choose between integrating the software with other secutiy mechanism or developing a substitute software.

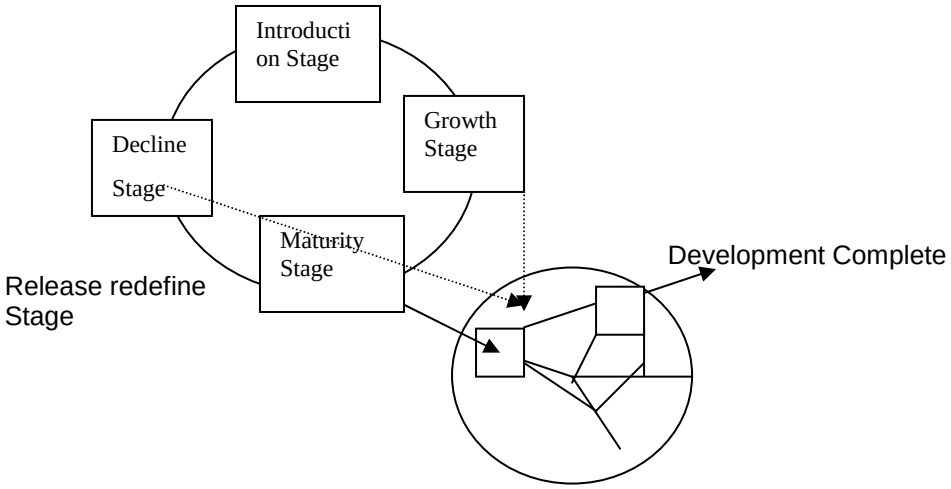

Figure 2. Characteristics of SSLC

3. Metrics Uses in OO Design:

Software Security is basically characterized by the four functions of software metrics, which are as follows:

i. Planning: Metrics are used to serve as the cost estimation, resources, training, scheduling, etc.
ii. Improving: Metrics used for the improvement of software quality.
iii. Controlling: Metrics used to size and schedule of the software process.
iv. Organizing: Metrics are used to check the status and track according to compliance.

Table 1. Metrics with their Threshold value

Metrics	Threshold Value
Average Size (LOC)	Should be less than 24 for C++ and 8 for Smalltalk.
Average Number of Instance variable/Class	Should be less than 6
Average Number of Methods/Class	Should be less than 20
Depth of Inheritance Tree (DIT)	Should be less than 10, starting from base class
Number of class/class Relationship in each module	Should be relatively high.
Number of Modules relationship	Should be low
Number of Instance Variables	Depends on group of methods in a class.
Average Number of comments Line per method	Should be greater than 1.
Number of problem per class	Should be low.
Number of Times class is Reused	If a class is not being reused in different application, it might need to be redesigned.
Number of classes and methods Thrown away	Should occur at a steady rate throughout most of the development process.
Fan In (FIN)	Should be low.
Public Access to Data member	Should be low.

For better understanding, we should analyze the relationship among these metrics and correlate them every three months after (snapshot). These are some metrics generally used in the time of Object Oriented software design phase, which play an important role during the security phase of the entire life cycle of the software system. The Table 1 shows the metrics with their Threshold value [4].

From the table shown above, some of these metrics are guidelines for Object Oriented design and development, while some of these metrics are used for quality indicator and a few is used for validation of Object Oriented development process.

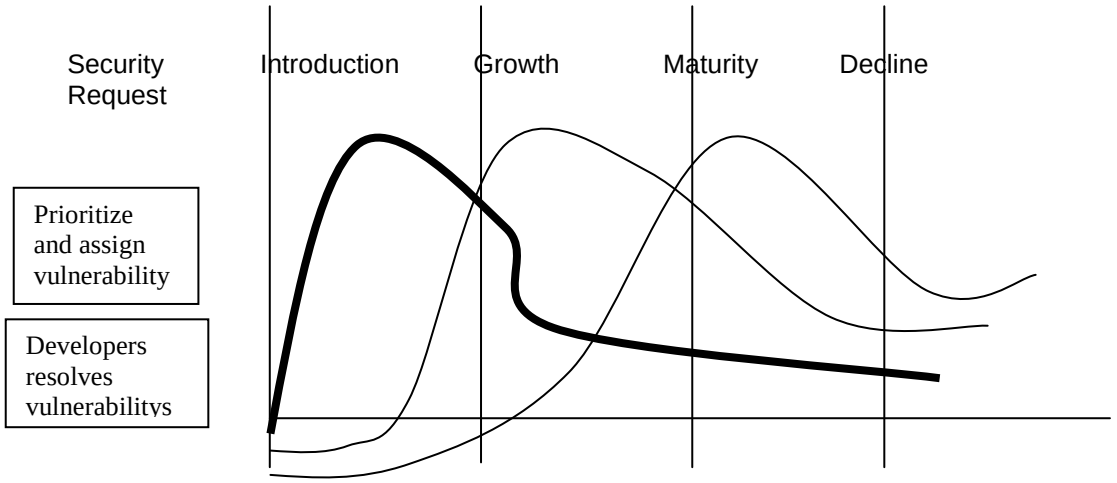

Figure 3. Stages of SSLC

4 Software Security Management Framework

The Software Security Life Cycle provides a general picture of the changes of software Security of Object Oriented software as shown in Figure 3. to apply the SMLC in software Security management as need to develop a framework providing quantifiable methods to identify different Security stage. The software Security management using Vulnerability Convergence Tracking process (Figure 1) consists of the following procedure such as Change Control, Testers, Vulnerability Traced, Prioritize and Vulnerability resolve.

5. Approach to Improve Software Security

In this paper, Microsoft Solution Framework (MSF) Process Model is used to improve the software quality and security to meet the acceptance criteria of a system. The testing phase design and document test specifications and test cases write automated scripts. And run acceptance tests as components that are submitted for formal testing.

In this phase, the team assesses and reports on overall solution quality and security and feature completeness. The testing team can trace vulnerabilities as shown in Figure 1 based on development and testing roles. The steps in the vulnerability tracking process are:

i) Developers develop code and then check this code into the change control system.
ii) Testers perform the daily or periodically build an external coverage testing on all submitted code.
iii) Testers submit vulnerabilities into vulnerability tracking system, entering vulnerabilities description and repeatability, severity and visibility variables.
iv) The development and test leads conduct a prioritization meeting and calculate each of the vulnerability priority.
   - Assign vulnerabilities that exceed the zero-defect criteria to developers for correction.
   - Omit vulnerabilities from a previous cycle of the sub-process that were corrected.
v) Assign developer's resolve or correct vulnerability.
vi) Developers perform internal coverage testing and then check this code into the change control system.

6. MSF Based Vulnerability Tracking And Categorizing Model

The MSF vulnerability categorization model builds on the risk categorization system. A risk is something that has not yet occurred, so probability of occurrence and impact of the risk to the project are appropriate variables to estimate. These two variables are multiplied together to produce the risk exposure value, which is used to prioritize these risks. The important variables to an individual vulnerability are:

Table 2. Prioritization Table.

Variable	Condition
Repeatability	A percentage in the range of 10% through 100%, where 100% shows that the vulnerability is reproducible on every test run.
Visibility	A percentage in the range of 10% to 100%, where 10% indicates that the vulnerability is visible under the most obscure conditions. Vulnerabilities that manifest themselves in environment with simple conditions are said to be highly visible.
Severity	An integer in the range of 1 through 10, where classification 10 vulnerabilitys presents the most impact to the solution or code.

Table 3. Big Prioritization Table.

Events	Repeatability	Visibility	Severity	Priority
1	1	1	10	20
2	.9	.9	9	16.2
3	.8	.8	8	12.8
4	.7	.7	7	9.8
5	.6	.6	6	7.2
6	.5	.5	5	5.0
7	.4	.4	4	3.2

The priority of the vulnerability can be calculated as:

(Repeatability + Visibility) X Severity = Priority

The vulnerabilities priority can be displayed by using prioritization matrix in Table 3.

7. Data Collection

The second step on the collection of priority data forms the software developer and tester. This data collection will form the core against which metrics values will be compared to develop the vulnerability-tracking model. The testing types can collect the data.

8. Coverage Testing

Coverage testing is low level technical testing. When a developer write a section of codes, expert creates an automated unattended installation and performs low level testing to ensure that the solution meets the functional specification. In MSF, it is also called as prefix testing. To perform such type of testing, a buddy testing involves using developers who are not working directly on the creation of particular code segment, and employing them to perform coverage testing on their colleague's code. This strategy works well because the skills, which are required by coverage testers, are on the same level as the skills that are required for development.

9. Usage Testing

Usage testing is potential users of the solution of this group and often performs a high level testing. This type of testing is very important because it ensures that vulnerabilities are related to user performance enhancement captured and addressed. Automated scripts and checklist are a best practice because they provide repeatability and perspective direction to the usage tester, which will improve accuracy.

10. Approach of Validation

This section discusses the validation of the estimated approach. As a matter of approaches validity, it must be verified that:

i) The data does not contain any vigorous data that violate any considered assumption for the development of the model. This can be achieved by plotting standardized residual values against standardized predicted value of the dependent variables. If the model is appropriate for data, the residuals should follow a normal distribution.

ii) Another validation plot is the standardized predicted values versus observed values to check the data coincidence. If the model fit each data value exactly, the observed and predicted values would coincide as straight line.

In our validation experiment 92 KLOC of C source code, taken from large and medium sized industrial system, the metric values were used in the model for assessing the security index. The difference between predicted value and the observed value of our validation data are close enough in most of the cases. One of the major difficulties in software Security is the Security of consistency of various documents, including requirement documents, design documents; comments in source code and source code themselves. MSF maintain the links and semantics of related sub model, which are represented, in standard models and documents. The consistency of these related documents can be enhanced through MSF, which then significantly help improve security.

11. Conclusion

Software Security is going to be a big challenge for many years to come. It is considered that predicting the software Security at design phase will definitely reduce the cost and effort of software development. The intent of this paper is to improve security using a vulnerability tracking and vulnerability removing approach. At the module level, this model can be used to

monitor changes to the system as they occur and as a method of predicting vulnerability prone model. Although this model may not be perfect in all environments, it demonstrates the utility of developing such models tailored for particular application domains. Developers and maintainers follow the process described in this paper to further customize the security model for their domain.

In this paper, we have presented a comprehensive approach to identify the metrics used in Object Oriented design and analysis to predict the security of a software system. The SMLC model and the management framework can identify the Security life cycle and synthesizes the product and integrate with Security.

Reference:

[1] Krishan K Aggarwal, Yogesh Singh, Jitender Kumar Chhabra. *An Integrated Measure of Software Manitainability*. Proceedings Annual Reliability and Security Symposium. 2002; 235-241.
[2] Jane Huffman Hayes, Sandip C. Patel, Liming Zhao. *A Metrics-Based Software Maintenance Effort Model*. Proceedings of the Eighth European Conference on Software Security and Reengineering (CSMR-04). 2004.
[3] Melis Dagpinar and Jens H. Jahnke. Predicting Security with Object Oriented Metrics- An Empirical Comparison. Proceedings *of the 10th Working Conf. On Reverse Engineering (WCRE-03*), 2003.
[4] Stephen H Kan. Metrics and Models in Software Quality Engineering. 2nd Edition. Pearson Education. 2003.
[5] Chu C William, I U Chih-wet, Yuehmin Huang, Boowen Xu. Software Security Improvement: Integrated standard and Models. *Proc. Of 28th Annual Intl. Computer Science and Application Conf. (COMPSAC02)*. 2002.
[6] Pankaj Jalote. An Integrated to Software Engineering. 2nd Edition. Narosa Publication Home. 2002.
[7] Roger Pressman. Software Engineering A Practitioner's Approach. 4th Edition. McGraw Hill Companies Inc. 1997.
[8] Microsoft.Net. Analyzing Requirement and Designing Microsoft.Net Solution Architecture. Prentice Hall India Pvt. Ltd. 2004.
[9] M HarrySneed, Agnes Kapose. A Study on the Effect of Reengineering Upon Software Security. IEEE Press. 1990; 91-99.
[10] Slaughter, S A. Software Development Practices and Software Security Performance. Thesis. Minnesota: University of Minnesota; 1995.