International Journal of Innovative Technology and Exploring Engineering (IJITEE)
ISSN: 2278-3075 (Online), Volume-8 Issue-11, September 2019

Framework for Testing the Security of Application Software at Design Phase

Neha Mahendra, Mohammad Muqeem

Abstract. ***Software security testing is essential to reveal the weaknesses in the security of the system. The security level of the software must be assessed properly and timely so that the security breaches can be prevented to occur otherwise they harm the system. Security testing during designing the software will be advantageous to reduce the rework and expenses required if it will be found insecure after the implementation. Security testing can be achieved efficiently through proper framework at the early stages of software development. Security can be checked at the initial level by taking inputs at the requirement phase and design phase so that loopholes can be found and the propagation of vulnerabilities can be prevented. At requirement phase security requirements can be filtered and then at the next phase designing artifacts can be inspected for security errors. A metric is designed which will grade the software under test and state that whether the system is secured at the proper level or not. In this paper a framework is proposed which is based on metric and the validation of the metric is done through the Weyuker's property.***

Keywords- Security testing framework, requirement phase, design phase, software development life cycle, security metric.

I.INTRODUCTION

The main objective of security testing is to check the weaknesses of the implemented security mechanism. Security testing process is a way to evolve software development process in a manner to reduce vulnerabilities. Security testing process gives the various security testing activities and tells us which activity is to be carried out in what sequence so that the appropriate security is achieved with minimum cost and effort. During all the phases of software development life cycle (SDLC), security should be the parallel thought while planning, designing and implementing functionalities in the system. Security testing must be performed in time before a breach harms the system. If a security breach is identified in the later stages of software development life cycle then it requires more effort and time to rectify it. A good security testing should incorporate the proper designing of the overall system. In the design phase, the skeleton of the software is prepared and the various artifacts are collected before stepping into the next stage of the SDLC. The application logics, algorithms, interface designs, database designs, process flows etc are available in the design phase. In the design phase, if the efficient approach is used then one can be capable of meeting all the security objectives prior to implementation.

Manuscript published on 30 September 2019.
***Correspondence Author(s)**
Mrs. Neha Mahendra, Ph.D. in Computer Application from Integral University, Lucknow, U.P., India.
Dr. Mohammad Muqeem, Associate Professor in the department of Computer Application, Integral University, Lucknow, UP, India.

Here the security risks, vulnerabilities, its impact and degree of impact can be well explained and mitigated. In the current scenario mostly the software is developed in the object oriented paradigm where the designing of the software are more focussed and weighted. Hence the characteristics of the object oriented paradigm should be taken into consideration in achieving the security of the software. A framework for testing the security at design phase of software development life cycle is proposed. The framework is composed of four stages which should be done sequentially. A security metric called as SMBC is designed and implemented in the framework for testing the software. SMBC provides the structure for designing any security metric and it is based on categorization of classes in different levels of security. Categorization of classes can be done according to the nature of the software under testing, platform used, need, etc. The proposed framework with the SMBC metric will provide the complete freedom for the implementation.

II.RELATED WORK

In the last one and half decade, various works have been performed for testing the security of a software system. Some of them also focused on integration of security with software development. During the Software Development Life Cycle (SDLC), the work performed for security testing is summarized. The focus is drawn on security prior to implementation. In 2002, K. Jiwnani and M. Zelkowitz [1] proposed a security testing strategy based on the three dimensional classification of vulnerabilities. Their genesis, location and impact are considered. This classification scheme fixes defects in the initial stages of the development cycle. In 2004, B. Potter, G. Mcgraw [2] proposed a risk-based approach to software security testing. They emphasized the importance of non functional security testing. Security problems can be solved before production of the software. In 2004, S. Lipner [3] proposed the security development lifecycle having many sub-processes distributed among all phases of SDLC. Threat modeling is placed on the top priority of software development life cycle.. In 2004, P. Bresciani, P. Giorgini, F. Giunchiglia, J. Mylopoulos, A. Perini[4] proposed a new agent oriented software development methodology 'Tropos' through which security can be implemented at design phase through a threat model. Also explained the five phases supported by Tropos. Diagrams are used to describe the model. In 2007, D. Byers and N. Shahmehri [5] presented a process consisting of vulnerabilities. These are based on vulnerability cause graphs. In 2007, S. Feruza, Prof. T. Kim [6] given a review based on Strategies and methods.

Different frameworks for assuring security in components are also discussed in detail. They also reviewed the technologies in IT security. In 2007, H. Mouratidis, Giorgini [7,8] proposed a Secure Tropos technique which is based on agents. Security Attack Scenarios are used with the Secure Tropos to validate the security solution with respect to the system's security requirements.

The use of secure Tropos for the security of information systems at design time is proposed and also given the applicability of the approach through a real-life case study. In 2008, I. A. Tondel, M. G. Jaatun and J. Jensen [9] presented a review of security testing and based on vulnerabilities in the organization proposed a software security testing scheme. The output of one application is the input to the next application to be tested. In 2008, P. R. Srivastava, et al [10] discussed on extension of Object-Oriented Software Testing Techniques to Agent Oriented Software Testing. For the generation of the test cases, random testing technique is used. In 2010, Z. Hui [11] presents a software security testing (SST) model based on Software Security Defects (SSD). The defects behavior analysis is performed with the help of SSD, software vulnerabilities, software security threats and accidents. In 2010, G. Tian-yang, S. Yin-sheng, and F. You-yuan [12] given a review based on major methods and security testing tools. Classification of the security testing into security functional testing and security vulnerability testing is presented. The taxonomy of security testing tools is given. In 2011, S. Jain, M. Ingle [13] reviewed the software metrics in software development process and suggests that there is still the scope of development of metrics for quantitative assessment of security. In 2012, S. J. Lincke, T. H. Knautz and M. D. Lowery [14] used the Misuse Deployment Diagram (MDD) for system architecture. It is based on UML. A case study on web registration project is performed. In 2012, H. Shahriar, M. Zulkernine [15] presented a review based on Security Vulnerability Mitigation Techniques. The comparison of various security vulnerability mitigation techniques with static analysis and hybrid analysis is given. Secure programming and patching are also discussed. In 2012, N. Sivakumar and K. Vivekanandan [16] compared the various approaches of building and testing the software with the help of a case study. The need for a specialized agent-oriented software testing mechanism is stated.

Later on the more focus is drawn on the design artifacts related to security testing process. Various models are designed for testing the security of the software system. Security testing techniques are implemented that spans in various phases of software development life cycle. The vulnerabilities deduction has been the centre point for testing the security of the system. Various techniques and metrics like Attribute Vulnerability Ratio (AVR) and Vulnerability Propagation (VP) are also used.

The various researchers put the plenty of efforts to integrate the security testing with software development process. But generally the techniques have been employed in the later stages. Hence, the development and implementation of the security needs a proper framework for achieving the more secure software on time. Hence various security testing approaches are proposed in the last decade but no proper framework exists for security testing at design phase [17]. As the prevention is better than cure, therefore design should be checked earlier for security rather than trying to secure already made software when threat occurs

III.DESIGN PHASE FRAMEWORK FOR SOFTWARE SECURITY TESTING

A framework is a detailed and complete structure for the realization of a specific task. According to the specific application and need, frameworks give the freedom for the implementation of the techniques defined. A good security framework should consider all the security features and examine the vulnerable areas of the software in a proper manner. In the current scenario, the characteristics of the object oriented paradigm should go through for the proper security of the software. Some of the features of the object oriented paradigm that affects the security of a software system at the design time are Abstraction, Encapsulation, Inheritance, Cohesion, Coupling, Complexity and Polymorphism [18, 19]. In order to quantify the affect of these properties on the software security, a security metric is needed [20, 21]. Considering the need for proper framework for security testing in design phase, the framework which is composed of four stages – Inception, Preparatory, Execution and Assessment is proposed (see figure-1).

In the software development life cycle, after the analysis and design phase, the security requirements and design will be given as input to the proposed framework. In the Inception stage, the information is taken from the requirement phase for security goals, security test items, etc.

Here the approach for security testing going to be adopted is decided and explained. In the second stage, all the preparations and data collection is done in reference to the security testing with the object oriented design of the software. In the third stage, the execution of the security testing procedures is done with the help of the identified software on the specific project selected for security testing. In the last stage, finalize the level of the security of the software tested. A security test closure report is prepared and documented for further references. In the figure colored blocks at each stage shows the output of that stage which goes to the next one.

At the end, the output from the framework will be a tested design. According to the outcome security grade one can proceed further to forthcoming stages of the software development life cycle if the security grade is satisfactory. If not satisfactory then rectification in the design should be done before implementation (see figure-2).

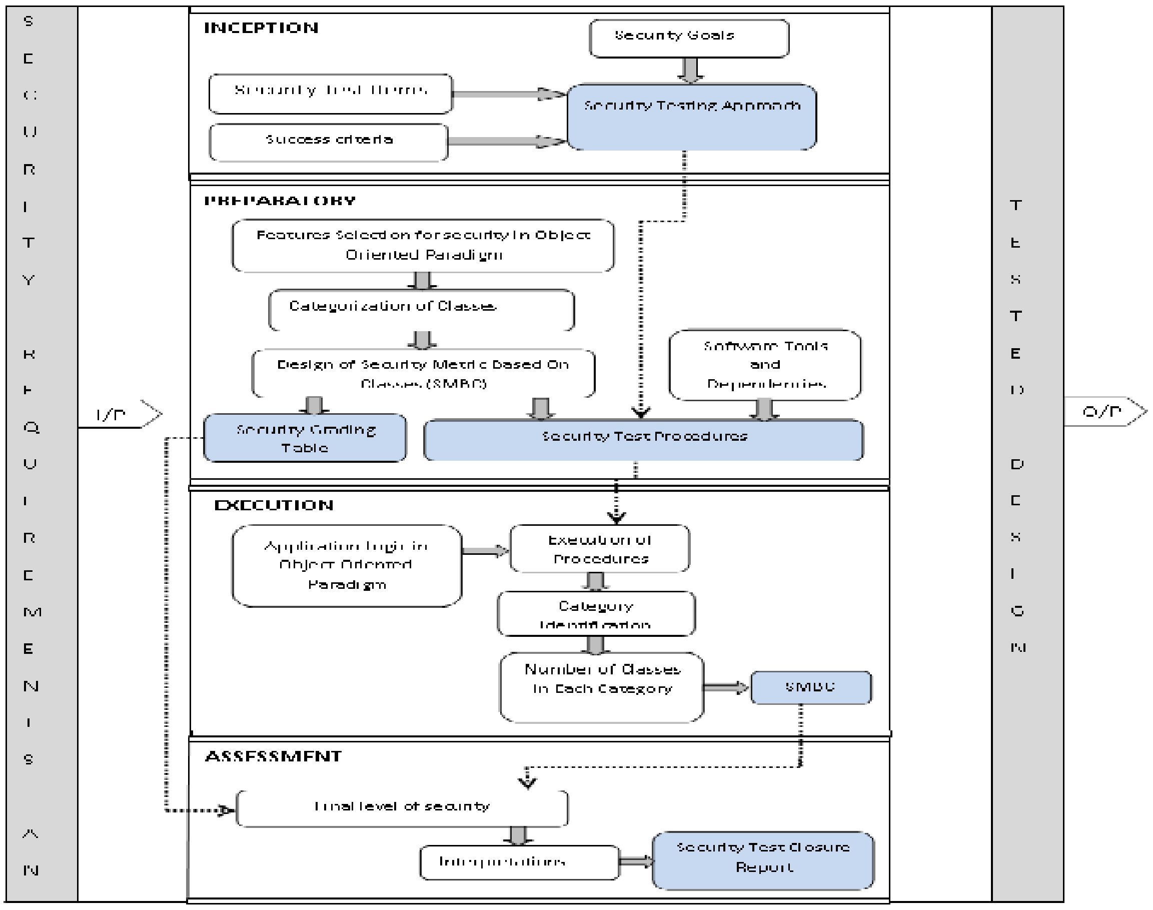
S E C U R I T Y R E Q U I R E M E N T S A N
I/P
INCEPTION
Security Goals
Security Test Items
Success criteria
Security Testing Approach
PREPARATORY
Features Selection for security in Object Oriented Paradigm
Categorization of Classes
Design of Security Metric Based On Classes (SMBC)
Software Tools and Dependencies
Security Grading Table
Security Test Procedures
EXECUTION
Application logic in Object Oriented Paradigm
Execution of Procedures
Category Identification
Number of Classes In Each Category
SMBC
ASSESSMENT
Final level of security
Interpretations
Security Test Closure Report
T E S T E D D E S I G N
O/P

Figure-1: Proposed Framework

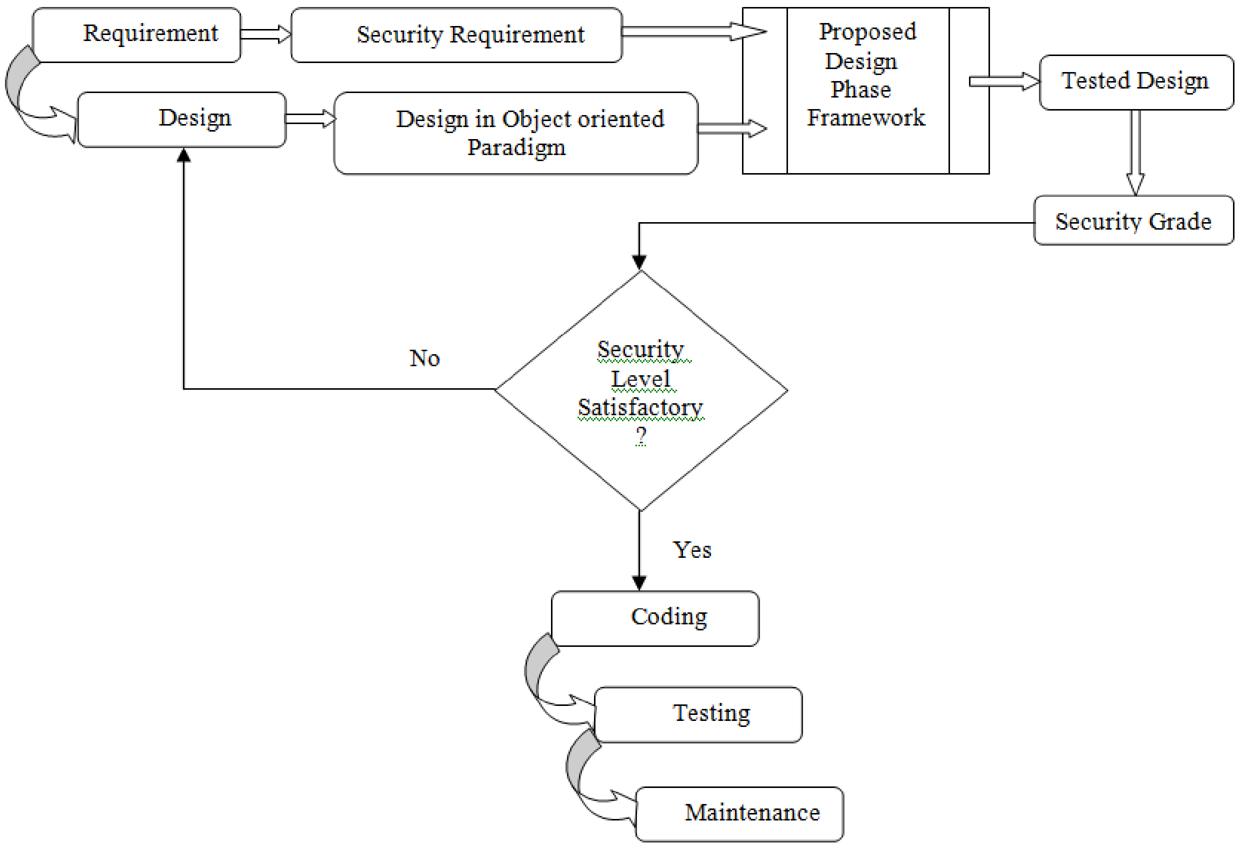
Requirement
Security Requirement
Design
Design in Object oriented Paradigm
Proposed Design Phase Framework
Tested Design
Security Grade
No
Security Level Satisfactory ?
Yes
Coding
Testing
Maintenance

Figure-2: Role of Proposed Framework in Software Development

A. Stage-1: Inception

This is the first stage and here the inputs for the security testing process are taken. The security goals and objectives must be clearly defined and explored. The basic objectives are the CIA of the security that is Confidentiality, Integrity and Availability. The items which are to be tested for security must be selected and listed. Both of these tasks are generally done in the requirement phase of the software development lifecycle [22]. Therefore, these inputs may be taken from the requirement phase database.

There should be criteria for judging whether the tested item is secure or not. State that criteria in reference to the approach used for security testing. Mention the overall approach going to be used for security testing explaining the rest of the stages in brief. At last, make a list of test deliverables that may include, Security test metrics, Reports, Charts etc.

```
Inception Stage Algorithm()
{
Security Goals(Requirement repository)
{
select security goals;
check feasibility;
filter and finalize the security goals;
}
//end Security Goals
Security Test Items(Requirement repository)
{
select security items;
find the related design material;
finalize security design items;
}
//end Security Test Items

Success Criteria()
{
design the success criteria for security;
convert in context of design;
}
//end Success Criteria

Security Testing Approach()
{
list the input requirements;
define security testing procedures;
define the domain of testing;
list the technical requirements;
validate the approach;
}
//end Security Testing Approach

}
//end Inception Stage Algorithm
```

B. Stage-2: Preparatory

This is the stage where all the necessary data has been collected and prepared for execution. A metric called as SMBC i.e. Security Metric Based on Classes is designed in this phase for testing the application logic or pseudo code or algorithm for getting the level of the security of the specific software. SMBC is based on the categorization of the classes in various levels of security that are designed according to the application. Security grades are decided according to the value of the SMBC. Software tools that are needed for testing are identified and installed. Assumptions and dependencies if any are listed and published to the security testing team. Security test procedures are written in accordance with the security testing approach and the software tools used.

First of all feature selection for security in Object Oriented Paradigm is done. Some of the features of the object oriented paradigm that affects the security of a software system at the design time are Abstraction, Encapsulation, Inheritance, Cohesion, Coupling, Complexity and Polymorphism. Abstraction is the feature of filtering the essential attributes from the real world problem. Encapsulation is the feature of binding the attributes and methods together and hiding them for the direct access from outside. Inheritance is the property of taking the parent attributes and methods and enhancing the reuse. Coupling refers to the extent with which different classes depend on each other and cohesion refers to the attributes and methods relatedness with each other within a single class. Complexity is the measure of the effort and time required for the work to be performed by the classes and objects. Polymorphism means one name many forms and provides the feature of referring multiple things in a same manner. Selection of the features for finding the security level should be done by a technical person who takes into consideration the proper measures for security testing. Next design the criteria to identify the category of classes and make a condition to belong to each category. The weightage of each class in security depends on the category to which it belongs. Higher the contribution in security leads the higher value of the weightage. The value of the weightage is taken in between 0 and 1.The following five categories of classes are considered as Highly-Secured, Secured, Moderate-Secured, Less-Secured and Un-Secured :-

- **Highly-Secured classes (HS) (weightage in security:C_1=1)**
  {Given condition for a class to belong in Highly-Secured category} $\longrightarrow$ condition-1
- **Secured classes (S) (weightage in security:C_2=0.75)**
  {Given condition for a class to belong in Secured category} $\longrightarrow$ condition-2
- **Moderate-Secured classes (MS) (weightage in security: C_3=0.50)**
  {Given condition for a class to belong in Moderate-Secured category} $\longrightarrow$ condition-3
- **Less-Secured classes (LS) (weightage in security: C_4=0.25)**
  {Given condition for a class to belong in Less-Secured category} $\longrightarrow$ condition-4
- **Un-Secured classes (US) (weightage in security: C_5=0)**

{Given condition for a class to belong in Un-Secured category} $\longrightarrow$ condition-5

For Highly-secured classes, the security is best that is why the value C_1 is taken as 1. In this case class contribution to security is maximum. It will maximize the value of the security metric as desired. For secured classes, the security is said to be good enough and the value of C_2 is taken as 0.75. For Moderate -Secured classes, the security achieved is average and the value of C_3 is taken as 0.50. For Less -Secured classes, the security is sacrificed but still better than unsecured and that is why the value of C_4 is normally set to 0.25. For Un-Secured classes, there is no security guaranteed that is why the value of C_5 is set to 0. However according to the nature of the application and the need of security the values of the C_1, C_2, C_3, C_4 and C_5 may be altered as to fit into to the security considerations.

A metric called as Security Metric Based on Classes (SMBC), which is based on the categorization of the classes in various levels of security according to the application is designed. In general SMBC is defined as-

$$\mathrm{SMBC} = \frac{\sum_i (\text{weightage[i]} \ * \ \text{number of classes in category[i]})}{\text{total number of classes}}$$

Or

$$\boxed{\mathrm{SMBC} = \left(\sum_{i=1}^{n} \mathrm{C_i} * CC_\mathrm{i}\right) / \ \mathrm{C_T}}$$

where

CC_i	= Number of classes in class category i (HS, S, MS, LS, US).
C_i	= Multiplier/Coefficient for the weightage of the security for class category i.
C_T	= Total Number of classes
n	= Total number of class categories

Now exploring the SMBC formula according to the categories defined in the above section, SMBC can be defined as-

$$\boxed{\mathrm{SMBC} = \frac{C_1 * CC_{HS} + C_2 * CC_S + C_3 * CC_{MS} + C_4 * CC_{LS} + C_5 * CC_{US}}{C_T}}$$

where

CC_{HS}	= Total Number of Highly-Secured classes
CC_S	= Total Number of Secured classes
CC_{MS}	= Total Number of Moderate-Secured classes
CC_{LS}	= Total Number of Less -Secured classes
CC_{US}	= Total Number of Un-Secured classes
C_T	= Total Number of classes

$(C_T = CC_{HS} + CC_S + CC_{MS} + CC_{LS} + CC_{US})$

C_1	= Multiplier for the weightage of Highly- Secured classes in security
C_2	= Multiplier for the weightage of Secured classes in security
C_3	= Multiplier for the weightage of Moderate - Secured classes in security

SMBC Range	**Security Grade**	**Security Level**	**Remark**
SMBC=1	O	Highest Security	System is said to be fully secured and safe.
0.75<=SMBC<1	A	Very High Security	System is said to be secured and safe at optimum level.
0.5<=SMBC<0.75	B	High Security	System is secured at satisfactory level and takes care of sensitive attributes.
0.25<=SMBC<0.5	C	Medium Security	System is less secured and satisfactory level is low.
0<SMBC<0.25	D	Low Security	System is not secured and security features need to be altered.
SMBC=0	E	Lowest Security	System is insecure and fully vulnerable and all the security features need to be redesigned to rectify it.

C_4	= Multiplier for the weightage of Less -Secured classes in security
C_5	= Multiplier for the weightage of Un-Secured classes in security

$0<=C_1<=1$, $0<=C_2<=1$, $0<=C_3<=1$, $0<=C_4<=1$, $0<=C_5<=1$ and $0<=SMBC<=1$.

The standard values for coefficients are taken as follows:

$C_1=1$, $C_2=0.75$, $C_3=0.50$, $C_4=0.25$ and $C_5=0$.

Higher the value of SMBC better is the security of the software. Security grades are decided according to the value of the SMBC**.** It is given and explained in table-1.

The calculation of the SMBC and the evaluation of the category count that is the number of classes in each category can be done with the code written in any language. Therefore, the Software and the tools that are needed for testing are identified and installed. Assumptions and dependencies if any are listed and published to the security testing team. For the categorization of the classes, the procedures for the selection criteria for any class to put it inside the declared categories should be designed and written. Decide the inputs and take application logic either in form of pseudo code, algorithm or program from the design repository. The calculation of the number of classes inside each category can be computerized and executed with the help of any selected software or tool.

```
Preparatory Stage Algorithm()
{
select security features in OOP;

Categorization of Classes()
{
define criteria for categorization;
make rule for each category;
list categories;
}
//end Categorization of Classes

Design SMBC()
{
decide weightage in security for each category;
SMBC= $(\sum_i(\text{weightage}[i] * \text{number of classes in category}[i])/$ total number of classes);
}
//end Design SMBC

Design security grading table()
{
decide the security grades;
assign the SMBC range for each grade;
write the comments and suggestions for each grade;
make final security grading table;
}
//end Design security grading table

install security testing tools;

list assumptions and dependencies;

design security test procedures(Design repository)
Table-1: Resultant Security Grading
{
decide computer language for execution;
decide inputs;
store all in a file;
convert in proper format;
write the code in selected language;
}
//end design security test procedures

}
//end Preparatory Stage Algorithm
```

C. Stage-3: Execution

After the first two stages, all the data is collected and prepared as needed. Now the execution of the proposed technique must be done step by step in a proper manner. First get the application logic in Object oriented paradigm from the database. Execute the security test procedures either using some tool or manually. Identify the category of each class using the categorization criteria defined in the previous stage. Count the number of classes in each category (CC_{HS}, CC_S, CC_{MS}, CC_{LS}, CC_{US}). Finally calculate the SMBC (Security Metric Based on Classes) using the designed formula.

```
Execution Stage Algorithm()
{
CategoryCount(Design repository)
{
//get Application Logic in OOP
        Read File Wordbyword("file path");

// category Identification
 while (there are words in the file)
        {
                for (each class in the file)
                {
                find(variables required for categorization of classes);
        }
//end for
     }
//end while

//count the number of classes in each category
     if (condition-1)
                $CC_{HS}$ ++;
        else if (condition-2)
                $CC_S$ ++;
     else if (condition-3)
                 $CC_{MS}$ ++;
     else if (condition-4)
                $CC_{LS}$ ++;
        else if (condition-5)
                $CC_{US}$ ++;
Calculate SMBC(); //function call
   }
//end CategoryCount
//SMBC calculation
 Calculate SMBC()
{
Initialize coefficients();//function call
SMBC = ((($C_1$* $CC_{HS}$)+($C_2$* $CC_S$)+ ($C_3$* $CC_{MS}$)+ ($C_4$* $CC_{LS}$)+ ($C_5$* $CC_{US}$))/total number of classes);
}
//end Calculate SMBC

Initialize coefficients()
{
        $C_1$=1;
      $C_2$=0.75;
        $C_3$=0.5;
      $C_4$=0.25;
      $C_5$=0;
}
//end Initialize coefficients
}
```

//end Execution Stage Algorithm

D. Stage-4: Assessment

Finally at last, record the SMBC value and refer the security grading table for finding the level of the security of the software tested. Now analyze the result and do interpretations accordingly. It should be concluded whether the security grade is satisfactory or not. If satisfactory then proceed to implementation and if not then rectification in the design should be done before implementation. A security test closure report will be prepared and well documented for future references.

```
Assessment Stage Algorithm()
{
finalize the security level; //refer to security grading table in preparatory stage

Make Interpretations()
{
describe the security testing result;
list effects;
}
//end Make Interpretations

Security Test Closure Report(Design repository)
  {
final comments;
give suggestions;
prepare report;
}
//end Security Test Closure Report

}
```

//end Assessment Stage Algorithm

E. Premises and Guidelines

The following premises have been considered while designing the proposed framework for security testing process at design phase:-

i. The requirement phase has been done and will provide the proper inputs to the design phase related to security testing process.
ii. The design phase also has been done and the design document is sufficient for achieving proper security testing of any software system.

The guidelines for proper realization of the proposed framework that should be considered during implementation are as follows:-

i. Identify all the security attributes properly in the very first stage i.e. Inception stage.
ii. While designing the security metric based on classes (SMBC), the value of the weightage coefficients may be altered according to the application and the features taken for the categorization of the classes.
iii. Security grading table should be designed in accordance with the need and level of the security demanded for the specific application.

IV. VALIDATION OF THE PROPOSED FRAMEWORK

Validation of any framework or metric can be done either theoretically or empirically. Theoretical validation can also be done in various ways like using Weyuker's properties, Kitchenham's properties or Briand's properties [23]. For the validation of the proposed framework, the Weyuker properties of measures are used. There are nine Weyuker's properties- Noncoarseness, Granularity, Nonuniqueness (Notion of Equivalence), Design Details are Important, Monotonicity, Nonequivalence of Interaction, Permutation, Renaming Property and Interaction Increases Complexity [24] that are needed to be satisfied for proving the validity. However, in the object oriented paradigm there are few properties that are mandatory and few properties that are needed to be hold for all types of metrics. Rest of the properties sometimes may violate depends on the type of object under validation. For the validation through Weyuker's properties let there are three application software X, Y and Z and a metric m. Weyuker's properties can be stated as follows:-

Property-1(Non-coarseness): $\exists X, Y: m(X) \neq m(Y)$. This implies that there should be some application software for which value of metric is different. This property must hold for all types of metrics [25].

Property-2(Granularity): If $m(X) = m(Y)$ then count (for such application software) < ∞. This implies that there should be a finite number of application software having the same metric value. This is an essential property for object oriented metric. This property must hold for all types of metrics [25].

Property-3(Non-uniqueness): There can exist distinct software application X and Y such that m (X) = m (Y). This implies that any two application software can have the same metric value.

Property-4(Design details are important): Given two application software X and Y, which provide the same functionality, does not imply $m(X) = m(Y)$. This means that doing the same work does not mean the same internal structure of the application software.

Property-5(Monotonicity): For all application software X and Y, if X+Y is the combination of X and Y then, m(X)≤m(X+Y) and m(Y)≤ m (X+Y) must hold. This implies that the metric for the combination of two application software can never be less than the metric for either of the component application software.

Property-6(Nonequivalence of Interaction): For application software X, Y, Z, if m (X) = m (Y) then it is not necessary that m (X+Z) = m (Y+Z) holds. This suggests that the resultant measurement of adding Z in X may be different from adding Z in Y.

Property-7(Permutation): This property states that permutation of elements within the item being measured can change the metric value. This property is applicable in traditional program design. It is not applicable for object oriented metrics.

Property-8(Renaming Property): If a class X is a renaming of a class Y, then m (X) = m (Y). The renaming property states that the name of entity has no effect on the metric value. That means if the name of any metric changed then its value will not be affected. This is an essential property for object oriented metric.

Property-9(Interaction Increases Complexity): X and Y are application software such that: m (X) + m (Y) < m (X+Y). The property states that when the two application software is combined, then the interaction between application software can increase the complexity metric value.

Now, taking each property one by one for the proposed framework as follows:-

Property-1(Non-coarseness): $\exists X, Y: m(X) \neq m(Y)$.

Let all the classes are Highly-secured in application software then m(X) can be calculated as:-

$$\mathrm{SMBC} = \frac{1 * \mathrm{C_T} + 0.75 * 0 \ + 0.50 * 0 + 0.25 * \ 0 + 0 * \ 0}{\mathrm{C_T}}$$

This implies m(X)=1 in the best case and the security is highest. This is the Best case (Highest Security, SMBC=1).

Let half of the classes are Highly- secured and half of the classes are secured in application software Y then m(Y) can be calculated as:-

$$\mathrm{SMBC} = \frac{1 * (\mathrm{C_T}/2) + 0.75 * \ (\mathrm{C_T}/2) + 0.50 * 0 + 0.25 * \ 0 + 0 * \ 0}{\mathrm{C_T}}$$

This implies m(Y)=0.875. It is the average case and the security is high.

Therefore, $\exists X, Y: m(X) \neq m(Y)$. Hence property-1 holds for the proposed framework.

Property-2(Granularity): If $m(X) = m(Y)$ then count (for such application software) < ∞.

As each software is unique and will have only the finite number of classes and classes also have only finite number of methods and variables, the different software which have same value of the SMBC will be finite in number. Hence property-2 holds for the proposed framework.

Property-3(Non-uniqueness): There can exist distinct software application X and Y such that m (X) = m (Y).

If half of the classes are Moderate- secured and half of the classes are Less-secured then

$$\mathrm{SMBC} = \frac{1 * 0 + 0.75 * 0 \ + 0.50 * (\mathrm{C_T}/2) + 0.25 * (\mathrm{C_T}/2) + 0 * \ 0}{\mathrm{C_T}}$$

This implies SMBC=0.375.

If half of the classes are Secured and half of the classes are Un-secured then

$$\mathrm{SMBC} = \frac{1 * 0 + 0.75 * (\mathrm{C_T}/2) \ + 0.50 * 0 + 0.25 * 0 + 0 * \ (\mathrm{C_T}/2)}{\mathrm{C_T}}$$

This implies SMBC=0.375.

Here the two softwares are different but the SMBC value of both is same. This implies that any two application software can have the same metric value. Hence property-3 holds for the proposed framework.

Property-4(Design details are important): Given two application software X and Y, which provide the same functionality, does not imply $m(X) = m(Y)$.

There may be two softwares, which have the same functionality but internally they may have different logic and algorithm. The classes inside may have different security levels, software may be designed using different technologies and different programming. So, the number of classes belongs to different categories defined in the proposed framework may vary for the similar types of software. Thus, it will give the different SMBC value for them. Hence property-4 holds for the proposed framework.

Property-5(Monotonicity): For all application software X and Y, if X+Y is the combination of X and Y then, m(X)≤m(X+Y) and m(Y)≤ m (X+Y) must hold. As the SMBC is like an averaging metric and when two application software are combined then it will give the value between the two SMBC values of the component software. Property-5 does not hold for the proposed framework. Weyuker says that property-5 does not hold for measures like effort and data flow [24]. Hence it is not an essential property and it may not hold for some metrics.

Property-6(Nonequivalence of Interaction): For application software X, Y, Z, if m (X) = m (Y) then it is not necessary that m (X+Z) = m (Y+Z) holds. To prove this property let there is an application software X in which half of the classes are Moderate- secured and half of the classes are Less-secured where total number of classes are 10 then SMBC value will be as follows:-

$$\mathrm{SMBC} = \frac{1 * 0 + 0.75 * 0 \ + 0.50 * 5 + 0.25 * 5 + 0 * \ 0}{10}$$

This implies SMBC (X) =0.375.

Let there is another application software Y in which half of the classes are Secured and half of the classes are Un-secured where total number of classes are 20 then SMBC value will be as follow:-

$$\mathrm{SMBC} = \frac{1 * 0 + 0.75 * 10 \ + 0.50 * 0 + 0.25 * 0 + 0 * \ 10}{20}$$

This implies SMBC (Y) =0.375.
Now, let there is another application software Z which has total 6 classes out of which half of the classes are Less-secured and half of the classes are Unsecured then

$$\text{SMBC} = \frac{1*0+0.75*\ 0+\ 0.50*0+0.25*3+0*\ 3}{6}$$

This implies SMBC (Z) =0.125.
Let Z is added to both software X and Y. Then the SMBC value of the (X+Z) can be calculated as follows:-

$$\text{SMBC} = \frac{1*0+0.75*\ 0+\ 0.50*5+0.25*(5\ +3)+0*\ 3}{16}$$

This implies SMBC (X+Z) =0.28125.
Similarly the SMBC value of the (Y+Z) can be calculated as follows:-

$$\text{SMBC} = \frac{1*0+0.75*\ 10+\ 0.50*0+0.25*3+0*\ (10\ +3)}{16}$$

This implies SMBC (Y+Z) =0.317307.
Hence, SMBC (X) = SMBC (Y) but SMBC (X+Z) ≠ SMBC (Y+Z). Therefore, property-6 holds for the proposed framework.

Property-7(Permutation): This property states that permutation of elements within the item being measured can change the metric value. This property is applicable in traditional program design. It is not applicable for object oriented metrics [23]. Hence, it is not applicable to the proposed framework.

Property-8(Renaming Property): If a class X is a renaming of a class Y, then m (X) = m (Y). The renaming property states that the name of entity has no effect on the metric value. This is an essential property for object oriented metric [23]. It is obvious that renaming the application software or class will not affect the category of the classes to which they belongs. Hence renaming will not affect the value of the SMBC. This implies that property-8 holds for the proposed framework.

Property-9(Interaction Increases Complexity): X and Y are application software such that: m (X) + m (Y) < m (X+Y). The property states that when the two application softwares are combined, then the interaction between application software can increase the complexity metric value. Property-9 does not hold for the proposed framework. Weyuker says that property-9 does not hold for measures like effort and data flow [24]. Hence it is not an essential property and it may not hold for some metrics.

Summary of the validation through Weyuker's properties has been stated in table-2.

Table-2: Summary of validation through Weyuker's property

Property	**Applicability of the Property**	**Status for the Proposed Framework**
Property 1	Must hold for all types of metrics [25]	Holds
Property 2	Must hold for all types of metrics [25] and must hold for object oriented metrics[23]	Holds
Property 3	Must hold for all types of metrics [25]	Holds
Property 4	Must hold for object oriented metrics [25]	Holds
Property 5	May not hold for some metrics[24]	Does not hold
Property 6	Must hold for object oriented metrics [25]	Holds
Property 7	Not applicable for object oriented metrics[23]	Not Applicable
Property 8	Must hold for object oriented metrics[23]	Holds
Property 9	May not hold for some metrics[24]	Does not hold

Thus, from table-2 it has been concluded that all the necessary properties hold for the proposed framework. Property 1, 2, 3, 4, 6, 8 must hold for OO metrics [25] and these all hold here. Hence the proposed framework has been validated.

V.CONCLUSION AND FUTURE SCOPE

Study reveals that there are sufficient artifacts at design phase of SDLC through which security testing can be assessed easily and timely. Some techniques are there that are applied at design phase but there does not exist any proper framework for security testing at design phase. Hence there seems a lack of robust framework for getting a proper security testing process which can be performed at design phase. After studying various papers in this context the complete framework composed of four stages is proposed. This can be considered as a generic framework through which security testing can be performed in a systematic manner so as to minimize the probable threats of a software system. For the realization of the framework after the identification of the proper security test items, only algorithm is needed. With the proposed framework, at the end of the design phase, the security flaws till far has been identified and can be removed before proceeding to further phases of software development. Hence the chances of the security breaches because of the flawful design will be neutralized and further propagation of vulnerabilities is prevented. In the further phases of software development life cycle, testing domain is narrowed and will be more focused only on the corresponding phase changes. In the proposed framework, to assess the level of security of any software only the algorithm is needed which is easily available at design phase. A security metric called as SMBC is designed which actually provides the base for designing any security metric and it does not depend on the implementation details.

ACKNOWLEDGEMENT

This work is acknowledged through the Research and Development Cell, Integral University, Lucknow with the research manuscript communication number (MCN): IU/R&D/2019-MCN000698.

REFERENCES

1. K. Jiwnani and M. Zelkowitz, "Maintaining software with a security perspective," Proceedings of International Conference on Software Maintenance, pp. 194-203, 2002.
2. B. Potter, G. Mcgraw, "Software Security Testing", IEEE Computer Society, October 2004.
3. S. Lipner, "The trustworthy computing security development lifecycle," Computer Security Applications Conference, pp. 2-13, 2004.
4. P. Bresciani, P. Giorgini, F. Giunchiglia, J. Mylopoulos A. Perini, "TROPOS: An Agent Oriented Software Development Methodology", Journal of of Autonomous Agents and Multi-Agent Systems, Kluwer Academic Publishers Volume 8, Issue 3, Pages 203 – 236, May 2004.
5. D. Byers and N. Shahmehri, "Design of a Process for Software Security," The Second International Conference on Availability, Reliability and Security, Vienna, pp. 301-309, 2007.
6. S. Feruza Y. and Prof.Tao-hoon Kim, "IT Security Review: Privacy, Protection, Access Control, Assurance and System Security", International Journal of Multimedia and Ubiquitous Engineering, Vol. 2, No. 2, April, 2007.
7. H. Mouratidis, Giorgini, "Secure Tropos: A Security-Oriented Extension of the Tropos methodology", International Journal of Software Engineering and Knowledge Engineering 17 (2) 285-309, 2007.
8. H. Mouratidisa, P. Giorgini, "Security Attack Testing (SAT)—testing the security of information systems at design time", Elsevier, Information Systems, Volume 32, Issue 8, Pages 1166-1183, December 2007.
9. I. A. Tondel, M. G. Jaatun and J. Jensen, "Learning from Software Security Testing," Software Testing Verification and Validation Workshop, ICSTW '08. IEEE International Conference on, Lillehammer, pp. 286-294, 2008.
10. P. Ranjan Srivastava, et al., "Extension of Object-Oriented Software Testing Techniques to Agent Oriented Software Testing", Journal of Object Technology (JOT), Vol 7, No. 8, Nov-Dec. 2008.
11. Z. Hui, S. Huang, Bin Hu and Yi Yao, "Software security testing based on typical SSD: A case study," 3rd International Conference on Advanced Computer Theory and Engineering (ICACTE), Chengdu, pp. V2-312-V2-316, 2010.
12. G. Tian-yang, S. Yin-sheng, and F. You-yuan, "Research on Software Security Testing", World Academy of Science, Engineering and Technology 69, 2010.
13. S. Jain, M. Ingle, "A Review of Security Metrics in Software Development Process", (IJCSIT) International Journal of Computer Science and Information Technologies, Vol. 2 (6), 2627-2631, ISSN: 0975-9646, 2011.
14. S. J. Lincke, T. H. Knautz and M. D. Lowery, "Designing System Security with UML Misuse Deployment Diagrams," Software Security and Reliability Companion (SERE-C), IEEE Sixth International Conference on, Gaithersburg, pp. 57-61.
15. H. Shahriar, M. Zulkernine, "Mitigating program security vulnerabilities: Approaches and challenges", ACM Computing Surveys (CSUR), ACM New York, NY, USA, Volume 44 Issue 3, Article No. 11, ISSN: 0360-0300 EISSN: 1557-7341, June 2012.
16. N. Sivakumar and K. Vivekanandan, "Exploring the need for specialized testing technique for an agent-based software, "International Journal of Soft Computing and Engineering (IJ SCE), vol. 2, pp. 221-228, 2012.
17. N. Mahendra and S. A. Khan, "A Categorized Review on Software Security Testing", International Journal of Computer Applications, 154(1):21-25, November 2016.
18. J. Bansiya and C. G. Devis, "A Hierarchical Model for Object Oriented Design Quality Assessment", IEEE transactions of Software Engineering, Vol 28, No. 1, January 2002.
19. Saarela, Marko & Hosseinzadeh, Shohreh & Hyrynsalmi, Sami & Leppänen, "Measuring Software Security from the Design of Software", 179-186. 10.1145/3134302.3134334, 2017.
20. K.Mustafa, R.A.Khan, "Quality Metric Development Framework (qMDF)", Journal of Computer Science 1 (3): 437-444, ISSN 1549-3636, 2005.
21. Thirugnanam, J.N. Swathi, "Quality Metrics Tool for Object Oriented Programming", International Journal of Computer Theory and Engineering, Vol. 2, No. 5, 1793-8201, October 2010.
22. N. Mahendra and Mohd. Muqeem, " An approach for the inception of security testing in the early stages of software development", Proceedings of the IEEE International conference on Computational and Characterization Techniques in Engineering and Sciences, Integral University, Lucknow, pp-51, September 14-15, 2018.
23. K.P. Srinivasan and T. Devi, "Software Metrics Validation Methodologies In Software Engineering", International Journal of Software Engineering & Applications (IJSEA), Vol.5, No.6, DOI : 10.5121/ijsea.2014.5606 87, November 2014.
24. Weyuker, Elaine," Evaluating software complexity measures", IEEE Transactions on Software Engineering, 14. 1357 - 1365. 10.1109/32.6178, 1988.
25. Misra, Sanjay & Akman, "Applicability of Weyuker's Properties on OO Metrics: Some Misunderstandings", Computer Science and Information system, 5. 17-23. 10.2298/CSIS0801017M, 2008.

AUTHORS' BIOGRAPHICAL SKETCH

Mrs. Neha Mahendra is pursuing a Ph.D. in Computer Application from Integral University, Lucknow, U.P., India. She is currently working as a Head Computers at Georgions academy Lucknow, UP, India. She worked as Assistant Professor in the department of Computer Application, Azad Institute of Engineering & Technology, Lucknow, UP, India for 14 years. Her research interests are in the areas of software security, security testing and software complexity. She is currently working in the area of software security testing. She is the member of IAENG.

Dr. Mohammad Muqeem has earned his doctoral degrees from Integral University, Lucknow. He is currently working as an Associate Professor in the department of Computer Application, Integral University, Lucknow, UP, India. Dr. Mohammad Muqeem is a young, energetic researcher. He has more than fourteen years of experience in the field of academics. He is currently working in the area of requirement engineering and software security. He is a member of CSI, ISTE, CSTA, IAENG and other societies.