Journal of Software Engineering and Applications, 2012, 5, 841-849
http://dx.doi.org/10.4236/jsea.2012.511097 Published Online November 2012 (http://www.SciRP.org/journal/jsea)

Testability Estimation of Framework Based Applications

Noopur Goel, Manjari Gupta

Department of Computer Science, Banaras Hindu University, Varanasi, India.
Email: noopurt11@gmail.com, manjari_gupta@rediffmail.com

Received August 29th, 2012; revised September 28th, 2012; accepted October 7th, 2012

ABSTRACT

Quality of software systems is highly demanded in today's scenario. Highly testable system enhances the reliability also. More than 50% of test effort-time and cost are used to produce a highly testable system. Thus, design-for-testability is needed to reduce the test effort. In order to enhance the quality, productivity and reduced cost of the software organizations are promoting to produce the reuse-oriented products. Incorporating reuse technology in both aspects-software development process and test process may payoff many folds. Keeping this view, our study focus the testability of the object-oriented framework based software systems and identify that flexibility at the variable points of the object-oriented framework, chosen for framework instantiation, greatly affects the testability of object-oriented framework based software at each level of testing. In the current paper, we propose a testability model considering the flexible aspect of the variable point to estimate testability in the early phase, requirement analysis phase, of development process of the framework based software. The proposed model helps to improve the testability of the software before the implementation starts thus reducing the overall development cost.

Keywords: Reuse-Technology; Variation-Points; Object-Oriented Framework Based Software; Basic Test Cases; Framework Interface Classes; Other Application Classes

1. Introduction

Quality assurance is highly demanded in any software industry. Researchers and practitioners are aspiring to achieve the goal with many techniques. Software reuse and software testing are the two promising techniques to enhance the quality of the software applications.

Framework is one of the promising technologies fostering reuse. A framework is the reusable design (the context) of a system or a subsystem stated by means of a set of abstract classes and the ways the objects of (subclasses of) those classes collaborates [1]. Being a reusable pre-implemented architecture, a framework is designed "abstract" and "incomplete" and is designed with predefined points of variability, known as hotspots, to be customized later at the time of framework reuse [2]. A hotspot contains default and empty interfaces, known as hook methods, to be implemented during customization. While preserving the original design, parts of the framework are extended or customized to build applications using frameworks. A hook is a point in the framework that is meant to be adapted in some way such as by filling in parameters or by creating subclasses [3]. Hook description [3] is used for many possible implementations of the Framework Interface Classes (*FIC*), shown in **Figure 1**, for developing applications in the application engineering stage [4].

Software testing is effective if it detects faults in the early life cycle of the project development. Software testing is performed with the intent of finding faults and is a vital and indispensable part of the software development process which itself is a highly time consuming and costly affair. Testing of framework based applications is not a trivial task as the frameworks are inherently complex in structure. The resources for testing are limited [6], hence to achieve effective testing, the applications are considered to be designed for testability. The testability of software is an important quality attribute, since it determines the effectiveness of testing, and hence the

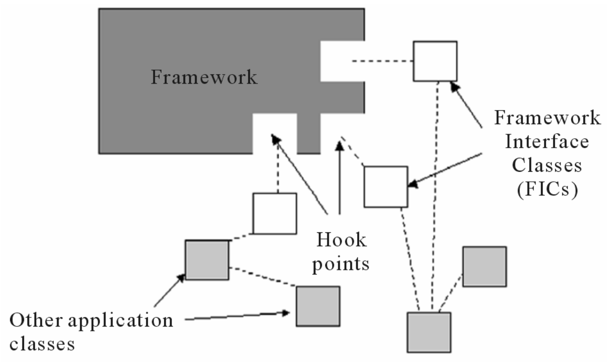


Figure 1. Framework Application Classes (hooks, framework interface classes, other application classes) [5].

probable correctness of the software. Testability is the external quality of the software [7] used to estimate the complexity and test effort. Bach *et al.* [8] define testability as effort needed for testing. Effort is the vital quality characteristic of the software and claims that the most important software characteristic contributing to testability is the number of test cases needed for satisfying a given test strategy.

Researchers have opined about software testability from many different aspects. In IEEE glossary [9], testability is defined as: 1) the degree to which the system or component facilitates the establishment of test criteria and performance of tests to determine whether those criteria have been met; 2) the degree to which a requirement is stated in terms that permit establishment of test criteria and performance of tests to determine whether those criteria have been met. ISO defines testability as the attributes of software that bear on the effort needed to validate the software product [10].

Flexibility provided at the hotspots of the framework promotes the reusability of framework. Flexibility is one of the internal quality attribute [6] associated to frameworks in addition to other software quality attributes: correctness, ease of use, efficiency, and portability. Increasing the flexibility of the framework not only increases the reuse opportunities of the framework, at the same time it increases the complexity also [11]. In this paper, we are investigating about "how flexibility affects the testability of the framework based application?"

In spite of the increasing practice of development of applications based on OO frameworks, as per our knowledge, no study is found in literature on the testability of applications based on OO frameworks. This paper identifies flexibility, the internal quality factor, affecting the testability of framework based application at all level of testing and proposes the testability model that considers the level of flexibility of hooks and test effort to reuse the basic test cases as-is, customize and/or generate new test cases to test the applications at application engineering stage.

The paper is organized in five sections. Although, as per our literature survey, no study is found on testability of OO frameworks based applications, related work on testability of OO software and OO frameworks, and testing of framework based application is discussed in Section 2. Section 3 describes motivations and objectives of testability study of OO frameworks based applications and contributions of the current paper. Background of our work is explained in Section 4. In Section 5, our proposed work is detailed. A case study to support the level of testing needed for various level of support of hook is performed in Section 6. Conclusion and future work is specified in Section 7.

2. Related Work

2.1. Software Testability Paradigm

Software testability has been defined and analyzed from different point of views by many researchers. Fenton [7] defines testability as an external attribute of the software. Binder [6] defines testability as the relative ease and expense of revealing software faults. According to him, a more testable system promotes high reliability of the software for a limited testing resource. Resources can be cost and time. He identifies six factors contributing to the design for testability: Characteristics of the representation, characteristics of the implementation, built-in-test capabilities, the test-suite, the test support environment and the software process in which testing is conducted. Design for testability and planning for test reduces the cost and difficulty of testing OO systems. Freedman [12] defines domain testability as the ease of modifying a program so that it is observable and controllable. It does not exhibit any test input-output inconsistencies. He identifies that a software component that is easily testable has the desirable attributes: test sets are small, test sets are non-redundant, test outputs are easily interpreted and software faults are easily locatable. [12,13] addressed testability using the concept of observability (ease of determining if specified inputs affect the outputs) and controllability (ease of producing a specified output from the specified input). Voas [13] defines testability as a probability that a test case will fail if a program has a fault. Testability metric based on input and output, and PIE (propagation, investigation and execution) approach to analyze software testability was proposed by Voas *et al.* [14,15]. Different from Voas *et al.*'s view [13-15], Bertolino *et al.* [16] proposed the program testability as the probability that a test of a program on an input drawn from a specified probability distribution of the given inputs is rejected, given a specified oracle and given that the program is faulty. Using testability they estimate, from test results: 1) the probability of program correctness; and 2) the probability of failures, and derives the probability of program correctness using the Bayesian inference procedure. Bruntnik *et al.* [17] estimated the testability of the class by distinguishing two categories of the source code: factors affecting the number of test cases needed to test it and factors affecting the effort required to develop each test case. Opposite to Bruntnik *et al.* [17] approach of unit testing of class, Jungmayr [18] estimated the testability in integration testing of the object oriented systems by providing the idea of the effect of static dependencies. The average component dependency metrics was taken into account to show how dependency between components can greatly affect the test effort of components integration and hence the testability of the system. Baudry *et al.* [19] proposes the methodology to improve the design testability by the UML class diagram of the system. They provide a model to capture the class interactions and identify classes that causes the interactions. Similar to concept of Baudry *et al.* [19], Mouchawrab *et al.* [20] also address the issues of object-oriented software testability at the design level *i.e.* before the implementation phase. Design attributes that affect testability for each testing activities are defined to provide the guidance for the testability analysis in the different testing phases.

2.2. Framework Testability Paradigm

Wang *et al.* [21] propose an approach of built-in-test (BIT) in object oriented framework which besides extending the reusability of frameworks from architecture to design to code to test improves the testability and maintainability of the frameworks. Similar to Wang *et al.* [21] approach, Jeon *et al.* [2] describes an approach to observe and control the process of framework testing, without making any changes to the framework design and code, whenever faults are incurred during the customization of the framework. A test support code is encapsulated as BIT component and embedded into the hook classes of the framework, increases the observability and controllability and hence testability of the framework. Jeon *et al.* [2] identifies four factors that have direct influence upon framework testability are: controllability, sensitivity, observability and oracle availability. Ranjan *et al.* propose various testability models based on abstractness/variability and design related aspects of the framework [22,23]. Ranjan *et al.* also carried an experimental analysis and found that study of testability of framework decreases with the increase in variability of the framework [24].

2.3. Framework Based Application Testing Paradigm

Framework testing at the framework engineering stage is important because the errors left in the framework during testing the framework during the domain engineering is propagated in the framework instantiation developed during application engineering stage. Several techniques have been proposed to test frameworks during the framework development stage (e.g., [25-33]). In [25,26] testing techniques, different possible framework use cases and input data are exercised. Tevanlinna *et al.* [33] observed that framework testing is hard during domain engineering stage. Some of the input data and use cases which are not covered at the time of framework testing in domain engineering stage and used in the framework instantiation, causes the instantiation to function improperly, must be located and tested during the application engineering stage. Dallal *et al.* [34] resolves the problems discussed in [33] by identifying the input data and use case not covered during the framework testing stage and proposing a test case reusing technique to test the uncovered input data and use cases thus reducing the testing effort. Kauppinen *et al.* [35] proposes hook and template based coverage criteria to test framework based applications.

2.4. Framework Based Application Testability Paradigm

As per our literature survey, no work on testability of framework based applications has been done. Although, there is much talk about increasing the quality and productivity while reducing the time-to-market of the software and software reuse practice is also prevalent but no study is found on the testability of framework based applications.

3. Motivations, Objectives and Contributions

Few obvious reasons for the need of testability study of applications based on OO frameworks which motivated us are discussed as follows:

1) Applications developed using reuse technology must be more testable in comparison to the development of application from the scratch. It must ensure low testing cost and hence low development cost of the OO frameworks based applications.

2) Reliability of the application increases with the increase in testability of the application.

3) Testability is important for maintainability of the application [ISO 9126].

The objective of the current paper is to deal with the issues mentioned above by identifying how flexibility affects the testability of the framework based application at each level of testing. Reuse of basic test cases to use as-is, customize and/or generate new test cases by the hook method specifications at the application engineering stage are considered as the effort taking test activities.

Our contributions to the current paper are:

1) Identification of flexibility as the important internal quality factor of the software affecting the testability of framework based application at each level of test.

2) Flexibility based testability model for the estimation of testability of framework based software before the implementation of software.

4. Background

In this section, we present the overview of the structure of testability models and then identify different levels of testing required for different level of support of the hooks. This forms the basis for the testability models proposed in this paper.

4.1. Test Techniques

Test techniques used to generate test cases from the specification of the software are known as specification based testing techniques. *FIC* specification in the hook description is used to derive test cases to test the OO frameworks. The test cases generated during the framework development stage are basic test cases. The basic test cases are reused to test the application developed using the OO framework during the application development stage rather than deriving test cases each time from the scratch. Thus, during the implementation of the *FIC*s, application developers deal with the specifications of *FIC*s described by the hooks in three ways [4]:

1) By using them as defined.

2) By ignoring the specifications for the behaviors that are not needed in implementing application requirements.

3) By adding new specifications for the added behaviors to meet application requirements.

Similarly, test cases generated using the hook method specifications are reused for developing the corresponding hook implementation in the following ways respectively:

1) By reusing them as-is.

2) By ignoring or modifying some of the reusable test cases.

3) By adding some more test cases or building new test cases from the scratch.

4.2. Levels of Testing

Object-oriented applications have three levels of testing: class testing, cluster testing and system testing. Class testing is almost same as unit testing and cluster testing is somewhat different but comparable to the level of integration testing of procedural approach. System testing is comparable in both approaches.

1) Class testing: Class testing is the first level of integration testing. The intra-class method interactions and super class/subclass interactions are carefully examined during the class testing level.

2) Cluster testing: The interclass collaborations and interactions between the system classes are performed at the cluster testing level. Interclass method interactions are taken into account. A cluster corresponds to a second level of integration. Cluster testing in framework based application is described in three ways—interaction of *FIC*s with framework classes, interaction between the *FIC*s and interaction of *FIC*s with other application classes.

3) System testing: Generally, the complete integrated system is tested based on acceptance testing requirements at the system testing level. All use cases defined at the analysis phase of the application engineering stage decides for the system testing of the application.

The object oriented application testing at method and system testing level is the same as traditional application software testing.

The specification based test cases for a class are produced by stating the specifications of each method and a class invariant. Verification of fulfillment to stated preconditions and post-conditions is performed by executing each method at least once. Other test cases may be derived based on the class invariants, if needed. Test cases for the intra and inter-class method interaction levels are derived for each pair of methods interacting with each other to detect which pair is responsible for producing faults. Functional test cases for system testing can be derived from the use cases and other system requirements.

4.3. Effort Related Test Activities

Test effort in case of framework based applications incurs due do the following activities:

1) Test units identification in framework based applications.

2) Organization and implementation of stubs and drivers for various units.

3) Identification of order of integration of units.

4) Finding and implementing test oracles.

5) Identification of reusable test cases and customization and generation of new test cases for the framework based applications for unit, integration and system testing at the application engineering stage.

6) Test case execution, analysis and debugging.

Test activities which needs intensive effort in context of our proposed approach is the identification of reusable test cases and customization and generation of new test cases for the framework based applications at the application engineering stage. Test cases for the framework based systems are constructed in two ways—either by reusing and customizing the baseline test cases or creating new test cases from the scratch by studying the required product.

4.4. Test Case Generation Techniques

1) Functional test cases are constructed by considering the specifications provided for implementing hook methods of the hotspots (abstract classes). State-based models are used to validate the post-conditions of the methods defined in the class.

2) Structural test cases are constructed by considering the implementation of each method.

3) Contracts [36] form the basis for the construction of interaction test cases as they are used for defining the interactions of classes within the cluster and thus act as oracles. Oracles are used to evaluate the actual results of the test cases as pass or fail.

5. Proposed Work

Frameworks are complex architectural skeleton and hence they also affect the testability of framework based systems. Because of the reusable nature of a framework, it must be made flexible to conform to its objective. Flexibility brings with itself complexity also. We identify that the variability existing at the hotspot of the framework are also of different level of flexibility. The different levels of flexibility are named as option, supported pattern and open hooks by Froehlich *et al.* [37]. Test effort to reuse basic test cases as, customize and/or generate new test cases from scratch at all levels of test at application engineering stage is studied and analyzed.

5.1. Level of Test to Be Conducted with Respect to Various Level of Support of Hooks for Framework Based Applications

There are various approaches of testing—testing product by product, incremental testing of product families and division of responsibilities [33]. For a hook documented framework, we have identified different levels of test already conducted during the framework engineering stage and required at the application engineering stage based on the level of support of hook. For each level of support depicted in the **Figures 2-4**, the gray area shows that the respective level of testing of the artifact is already conducted during the framework engineering stage and need no repetitions during the application engineering stage. If there exist other application classes, unit testing of them is needed and if there exist any interaction between the *FIC* and other application classes then integration testing is also needed. Only the trivial case of *FIC* consisting of hook with only one level of support *i.e.* either option or supported pattern or open hook is considered and depicted here.

Option Hook: Since the options are available within the framework, option hooks are already unit tested. However, when implementing option hooks, its implementations need unit testing during the application engineering stage. Further, interactions between option hooks as well as interaction between option hook and framework classes, if any, are also tested during framework engineering stage. Thus, integration testing is also not required in this case [37]. System testing of the application is required be conducted during the application engineering stage to test whether all functional requirements are fulfilled.

Supported Pattern Hook: Test cases can be generated which cover the range of parameters that the framework developers supply, are then adapted to test the applications. The templates should not be able to violate invariants on the framework, so no verification is needed [37]. Since the template pattern hooks should not violate

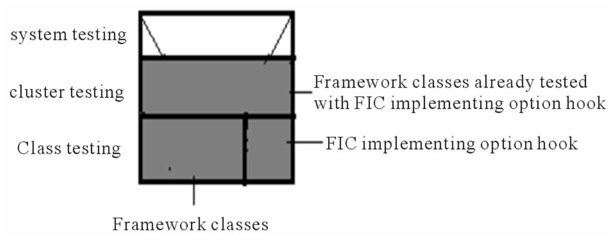
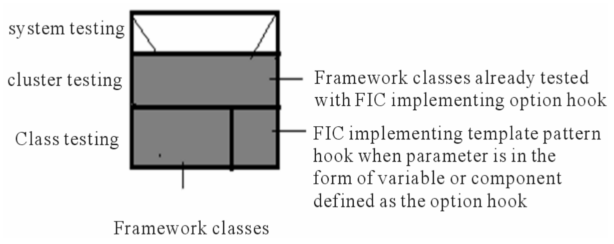
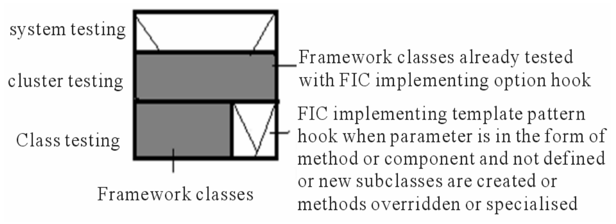

system testing
cluster testing
Class testing
Framework classes already tested with FIC implementing option hook
FIC implementing option hook
Framework classes

Figure 2. Various levels of test for *FIC* implementing option hook.


system testing
cluster testing
Class testing
Framework classes already tested with FIC implementing option hook
FIC implementing template pattern hook when parameter is in the form of variable or component defined as the option hook
Framework classes

(a)


system testing
cluster testing
Class testing
Framework classes already tested with FIC implementing option hook
FIC implementing template pattern hook when parameter is in the form of method or component and not defined or new subclasses are created or methods overridden or specialised
Framework classes

(b)

Figure 3. (a) Various levels of test for *FIC* implementing supported pattern hook when parameter is in the form of variable or component and stated as option hook; (b) Various levels of test for *FIC* implementing supported pattern hook when parameter is in the form of variable or component are not stated as option hook or new subclasses are created or methods are overridden or specialized.

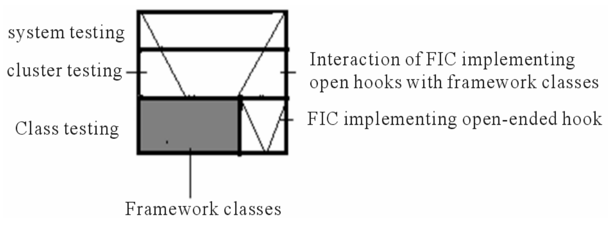

system testing
cluster testing
Class testing
Interaction of FIC implementing open hooks with framework classes
FIC implementing open-ended hook
Framework classes

Figure 4. Various levels of test for *FIC* implementing open hook.

the *FIC* method specifications defined in the hook description, the cluster testing *i.e.* the inter-class interactions, need not be performed at the time of application testing. The user is given some flexibility, so various cases may exist. In case (a) since the parameter is in the form of variable, method or component class which is already defined as option hook, so the class and cluster level testing is not required to be performed and only system testing is required to be performed during the application testing, as depicted in **Figure 3(a)**. Another case 3(b) is when *FIC* implementing pattern hook whose parameters are not defined as the option hooks and/or creation of new subclasses, or methods to be overridden or specialized, then the unit testing of the respective *FIC* and system testing is need to be conducted. The integration testing of the respective *FIC* with framework classes need not to be tested as depicted in **Figure 3(b)**.

Open Hook: Automated testing is not possible, and verification is considered necessary, maybe using model checking, to guarantee that the developers do not break conditions placed upon the framework or try to avoid or break the architecture of the framework [37]. New classes which are not subclasses, the new operations to classes and often the removal or replacement of code are made besides what is performed in the pattern hook.

The framework classes are already unit tested before the deployment of the framework. The framework interface class, *FIC*, having open ended hook are required to be unit tested, cluster testing is desired to be performed among the framework classes and *FIC* having open hooks, and system testing is also desired to be performed against the specified requirements of the application during the application testing.

Note: The gray area in the **Figures 2-4** how's the level of test already accomplished at the framework engineering stage and the white area shows the level of test required at the application engineering stage.

5.2. Comparison of Effort Needed for Various Levels of Test with Respect to Each Level of Support of Hooks

The flexibility to reuse as-is or customize basic test cases or generate new test cases from scratch during the application engineering stage increases from option hook to supported pattern hook to open hook [38]. It is obvious from the **Figures 2-4** that the effort needed for unit, integration and system testing is minimum with option hooks while it is maximum for open hooks. Thus, with the increase in flexibility of hooks test effort increases while the testability reduces as shown in **Table 1**.

5.3. Testability Model Considering the Flexibility of Hooks

The application developer chooses those hooks which satisfies the requirements of the specific application. The various levels of support of hooks are categorized on the basis of level of flexibility provided to the framework reuser to customize the framework. Requirements specified at the hook methods form the basis for the construction of test cases which can be reused during application testing. Thus test effort will depend on different level of support of hooks of a framework and hence on flexibility

Table 1. Comparative study of effort needed for various levels of testing with respect to each level of support of hooks.

Level of support of hooks / Level of testing	Option hook	Supported pattern hook	Open hook
Class testing	May not be needed	May/may not be needed	May be needed
Cluster testing	May not be needed	May not be needed	May be needed
System testing	May be needed	May be needed	May be needed

$\longrightarrow$

Effort in test activity at each level of testing increases with the increase in flexibility of hooks.

provided by hooks.

$$TE_A \alpha FL_A \quad (1)$$

Equation (1) is further refined below.

In case of option hooks, since all requirements are met, least or no flexibility is provided to the application developer to customize the requirements. Thus, flexibility provided by option hooks, FL_{OH}, at the variation point is least and proportional to the total number of option hook, N_{OH}, chosen by the application developer, *i.e.*

$$FL_{OH} \alpha \ N_{OH} \quad (2a)$$

Further, the components are developed, tested and delivered with the framework, so least or negligible amount of test effort is desired to be performed. Hence, test effort for the option hooks, TE_{OH}, depends on the flexibility provided by option hooks, FL_{OH}. Thus,

$$\sum_{i=1}^{N_{FIC}} TE_{OH} \alpha \ FL_{OH} \quad (2b)$$

In case of supported pattern hook, application developers are given the flexibility, FL_{SPH}, to reuse the specifications of the supported hook as-is or customize them. Flexibility provided by supported pattern hooks, FL_{SPH}, at the variation point is greater than FL_{OH} and is proportional to the total number of supported pattern hook, N_{SPH}, chosen by the application developer, *i.e.*,

$$FL_{SPH} \alpha \ N_{SPH} \quad (3a)$$

Further, the test effort depends on the number of basic test cases are reused as-is or customized as per the requirements specified at hook method are met or customized. Test effort for the supported pattern hooks, $TE_{SPH,}$ depends on the flexibility, FL_{SPH}, of supported hook in a *FIC*. Thus,

$$\sum_{i=1}^{N_{FIC}} TE_{SPH} \alpha \ FL_{SPH} \quad (3b)$$

In case of open hooks, application developers are given the highest flexibility to reuse the specifications of the open hook as-is or customize or generate new specifications from scratch. Flexibility provided by open hooks, FL_{OPH}, at the variation point/hotspot is highest and is proportional to the total number of open hooks, N_{OPH}, chosen by the application developer, *i.e.*,

$$FL_{OPH} \alpha\ N_{OPH} \tag{4a}$$

Further, test effort depends on the number of basic test cases reused as-is, customized or new test cases are constructed from scratch as per the requirement specified at hook are met, customized or specified by the application developer. Test effort for the open hooks depends on the flexibility of open hooks, FL_{OPH}, in a *FIC*. Thus,

$$\sum_{i=1}^{N_{FIC}} TE_{OPH} \alpha FL_{OPH} \tag{4b}$$

Thus, total flexibility, FL_A, chosen by the application developer provided at the variation point/hotspot of the framework is represented as

$$FL_A = FL_{OH} + FL_{SPH} + FL_{OPH} \tag{5a}$$

Since a *FIC* consists of option, supported pattern and open hooks, test effort for a *FIC*, TE_{FIC}, is proportional to the summation of the test effort of option, supported pattern and open hooks in a *FIC*, *i.e.* TE_{OH}, TE_{SPH} and TE_{OPH} respectively, or test effort for all *FIC*s is represented as

$$\sum_{i=1}^{N_{FIC}} TE_{FIC} \alpha \sum_{i=1}^{N_{FIC}} TE_{OH} + \sum_{i=1}^{N_{FIC}} TE_{SPH} + \sum_{i=1}^{N_{FIC}} TE_{OPH} \tag{5b}$$

Using Equations (2b), (3b) and (4b), Equation (5b) becomes

$$\sum_{i=1}^{N_{FIC}} TE_{FIC} \alpha\ FL_{OH} + FL_{SPH} + FL_{OPH} \tag{5c}$$

Using Equuation (5a), Equation (5c) becomes

$$\sum_{i=1}^{N_{FIC}} TE_{FIC} \alpha\ FL_A \tag{5d}$$

Test effort required to test the framework based application, TE_A, is dependent on the total test effort for total number of *FIC*s, N_{FIC}, containing hook(s) with various level of support and total test effort the total number of other application classes (N_{OAC}). Thus,

$$TE_A = \sum_{i=1}^{N_{FIC}} TE_{FIC} + \sum_{i=0}^{N_{OAC}} TE_{OAC} \tag{6}$$

So, replacing $\sum_{i=0}^{N_{OAC}} TE_{OAC}$ by the constant *c*, we have

$$TE_A \alpha \sum_{i=1}^{N_{FIC}} TE_{FIC} \tag{7}$$

Using Equation (5d), Equuation (7) becomes

$$TE_A \alpha\ FL_A \tag{8}$$

And using Equation (5b), Equation (7) is equivalent to

$$TE_A \alpha \sum_{i=1}^{N_{FIC}} TE_{OH} + \sum_{i=1}^{N_{FIC}} TE_{SPH} + \sum_{i=1}^{N_{FIC}} TE_{OPH} \tag{9}$$

Factually, the testability of application, Tb_A is inversely proportional to test effort, TE_A. Thus,

$$Tb_A \alpha \frac{1}{\sum_{i=1}^{N_{FIC}} N_{OH} + \sum_{i=1}^{N_{FIC}} N_{SPH} + \sum_{i=1}^{N_{FIC}} N_{OPH}} \tag{10}$$

The Equation (1) shows that test effort of a framework based application is directly proportional to the flexibility provided by the framework developer and chosen and implemented by the framework reuser as the specific needs. Equation (10) is the testability model of the framework based application dependent on the number of various level of support of hooks implemented in the total *FIC*s.

6. Case Study

A case study is performed for various levels of testing with respect to each level of support of hooks to explain the concept discussed in Subsection 5.1 by considering the hook examples of *HotDraw* framework discussed in [37]. First, the case of option hooks is considered. Select Existing Tools is a multi-option hook which application developer selects from the set of pre-built components as per the needs. Such components are already unit tested. The interactions of such components are also already tested with the framework classes. What one needs to perform is the system testing during the application engineering stage to verify that all functional requirements are met as desired. Second, the case of supported pattern hooks is considered. Incorporate Tools is a supported pattern hook. The changes section of the hook describes the steps needed to perform the task. Subclass NewDrawingEditor of framework class DrawingEditor is created and NewDrawingEditor.defaultTools overrides DrawingEditor.defaultTools features. The unit testing of NewDrawingEditor subclass is needed, interaction testing of subclass NewDrawingEditor with framework class DrawingEditor is not required to be conducted as the subclass NewDrawingEditor is created according to the contracts specified by DrawingEditor class. System testing is needed to be performed in this case during the application engineering stage to verify that all functional requirements are met as desired. Third, the case of open hooks is considered. The Animating Figures hook is an open hook and the changes section describes that AnimDrawing and AnimFigure are subclasses of Drawing and Figure. AnimDrawing overrides the existing step method of Drawing. These subclasses must be unit tested during the application engineering stage. The interaction of these subclasses AnimDrawing and AnimFigure must be tested with the framework classes Drawing and Figure. System testing is needed to be performed in this case during the application engineering stage to verify that all functional requirements are met as desired.

7. Conclusions

Our work, in the current paper, focus on identifying the internal software quality factor—flexibility affecting the testability of framework based software at all levels of testing. The level of flexibility provided at the variation point of the framework and chosen by the reuser of the framework as per the requirements affects the testability of the framework based systems. Moreover, we also propose a testability model considering the flexible aspect of the variable point to estimate testability in the early phase, requirement analysis phase, of development process of the framework based software. The proposed model helps to improve the testability of the software before the implementation starts thus reducing the overall development cost. The limitation of our study is that the observations are based on the hook documented framework.

In future, a number of extensions to our work is planned to be done:

1) An empirical study is needed to be conducted for the proposed model.

2) Modifying and proposing a new UML profile for the framework and framework instantiation to model the level of support of hooks, so that testability can be estimated directly from it.

3) Identifying and modeling many more factors affecting the testability of frameworks and framework based systems.

REFERENCES

[1] K. Beck and R. Johnson, “Patterns Generate Architectures,” *Proceedings of* 8*th European Conference on Object Oriented Programming*, Bologna, 1994, pp. 139-149.

[2] T. Jeon, S. Lee and H. Seung, “Increasing the Testability of Object-Oriented Frameworks with Built-In Test,” *Lecture Notes in Computer Science*, Vol. 2402, 2002, pp. 873-881.

[3] G. Froehlich, H. J. Hoover, L. Liu and P. Sorenson, “Hooking into Object-Oriented Application Frameworks,” *Proceedings of the* 19*th International Conference on Software Engineering*, Boston, May 1997, pp. 491-501.

[4] J. Al Dallal, “Class-Based Testing of Object-Oriented Framework Interface Classes,” Ph.D. Thesis, Department of Computing Science, University of Alberta, 2003.

[5] J. Al Dallal and P. Sorenson, “Estimating the Coverage of the Framework Application Reusable Cluster-Based Test Cases,” *Information and Software Technology*, Vol. 50, No. 6, 2008, pp. 595-604. doi:10.1016/j.infsof.2007.07.006

[6] R. V. Binder, “Design for Testability in Object-Oriented Systems,” *Communications of the ACM*, Vol. 37, No. 9, 1994, pp. 87-101. doi:10.1145/182987.184077

[7] N. Fenton and S. L. Pfleeger, “Software Metrics: A Rigorous and Practical Approach,” PWS Publishing Company, Boston, 1997.

[8] R. Bache and M. Mullerburg, “Measure of Testability as a Basis for Quality Assurance,” *Software Engineering Journal*, Vol. 5, No. 2, 1990, pp. 86-92. doi:10.1049/sej.1990.0011

[9] IEEE, “IEEE Standard Glossary of Software Engineering Terminology,” IEEE CSP, New York, 1990.

[10] ISO/IEC 9126, “Software Engineering Product Quality,” 1991.

[11] W. Pree and H. Sikora, “Design Patterns for Object Oriented Software Development,” *ICSE*’97, *Proceedings of the* 19*th International Conference on Software Engineering*, Boston, 1997, pp. 663-664.

[12] R. S. Freedman, “Testability of Software Components,” *IEEE Transactions on Software Engineering*, Vol. 17, No. 6, 1991, pp. 553-564.

[13] J. M. Voas, “Pie: A Dynamic Failure-Based Technique,” *IEEE Transactions on Software Engineering*, Vol. 18, No. 8, 1992, pp. 717-727. doi:10.1109/32.153381

[14] J. Voas and K. W. Miller, “Semantic Metrics for Software Testability,” *Journal of Systems and Software*, Vol. 20, No. 3, 1993, pp. 207-216. doi:10.1016/0164-1212(93)90064-5

[15] J. M. Voas and K. W. Miller, “Software Testability: The New Verification,” *IEEE Software*, Vol. 12, No. 3, 1995, pp. 17-28. doi:10.1109/52.382180

[16] A. Bertolino and L. Strigini, “On the Use of Testability Measures for Dependability Assessment,” *IEEE Transactions on Software Engineering*, Vol. 22, No. 2, 1996, pp. 97-108. doi:10.1109/32.485220

[17] M. Bruntink and A. V. Deursen, “Predicting Class Testability Using Object-Oriented Metrics,” *Proceedings of IEEE International Workshop on Source Code Analysis and Manipulation*, Chicago, 15-16 September 2004, pp. 136-145. doi:10.1109/SCAM.2004.15

[18] S. Jungmayr, “Identifying Test-Critical Dependencies,” *Proceedings of IEEE International Conference on Software Maintenance*, Montreal, 3-6 October 2002, pp. 404-413.

[19] B. Baudry, Y. L. Tran and G. Sunye, “Measuring Design Testability of a UML Class Diagram,” *Information and Software Technology*, Vol. 47, No. 1, 2005, pp. 859-879. doi:10.1016/j.infsof.2005.01.006

[20] S. Mouchawrab, L. C. Briand and Y. Labiche, “A Measurement Framework for Object-Oriented Software Test-

ability," *Information and Software Technology*, Vol. 47, No. 1, 2005, pp. 979-997. doi:10.1016/j.infsof.2005.09.003

[21] Y. Wang, D. Patel, G. King, I. Court, G. Staples, M. Ross and M. Fayad, "On Built-In Test Reuse in Object-Oriented Framework Design," *ACM Computing Surveys*, Vol. 32, No. 1, 2000, pp. 7-12. doi:10.1145/351936.351943

[22] D. Ranjan and A. K. Tripathi, "Variability-Based Models for Testability Analysis of Frameworks," *Journal of Software Engineering and Applications*, Vol. 3, No. 5, 2010, pp. 455-459. doi:10.4236/jsea.2010.35051

[23] D. Ranjan and A. K. Tripathi, "Testability Models for Object-Oriented Frameworks," *Journal of Software Engineering and Applications*, Vol. 3, No. 6, 2010, pp. 536-540. doi:10.4236/jsea.2010.36061

[24] D. Ranjan and A. K. Tripathi, "Effect of Variability of a Framework upon Its Testing Effort: An Empirical Evaluation," 5*th International Conference on Computer Sciences and Convergence Information Technology*, Seoul, 30 November-2 December 2010, pp. 146-151. doi:10.1109/ICCIT.2010.5711046

[25] R. V. Binder, "Testing Object-Oriented Systems: Models, Patterns, and Tools," Addison-Wesley Professional, Boston, 1999.

[26] J. Al Dallal and P. Sorenson, "System Testing for Object-Oriented Frameworks Using Hook Technology," *Proceedings of the* 17*th IEEE International Conference on Automated Software Engineering*, Edinburgh, September 2002, pp. 231-236.

[27] M. E. Fayad and D. C. Schmidt, "Object-Oriented Application Frameworks," *Communications of the ACM*, Vol. 40, No. 10, 1997, pp. 32-38. doi:10.1145/262793.262798

[28] T. Jeon, H. W. Seung and S. Lee, "Embedding Built-In Tests in Hot Spots of an Object-Oriented Framework," *ACM Sigplan Notices*, Vol. 37, No. 8, 2002, pp. 25-34. doi:10.1145/596992.597001

[29] J. Al Dallal and P. Sorenson, "Reusing Class-Based Test Cases for Testing Object-Oriented Framework Interface Classes," *Journal of Software Maintenance and Evolution: Research and Practice*, Vol. 17, No. 3, 2005, pp. 169-196. doi:10.1002/smr.308

[30] W. Tsai, Y. Tu, W. Shao and E. Ebner, "Testing Extensible Design Patterns in Object-Oriented Frameworks through Scenario Templates," *Proceeding of* 23*rd Annual International Computer Software and Applications Conference*, Phoenix, October 1999, pp. 166-171.

[31] M. E. Fayad, Y. Wang and G. King, "Built-In Test Reuse," In: M. E. Fayad, Ed., *The Building Application Frameworks*, John Wiley and Sons, Chichester, 1999, pp. 488-491.

[32] E. Gamma, R. Helm, R. Johnson and J. M. Vlissides, "Design Patterns: Elements of Reusable Object-Oriented Software," Addison-Wesley Professional Computing Series, Boston, 1994.

[33] A. Tevanlinna, J. Taina and R. Kauppinen, "Product Family Testing: A Survey," *ACM Sigsoft Software Engineering Notes*, Vol. 29, No. 2, 2004, pp. 12-18. doi:10.1145/979743.979766

[34] J. Al Dallal and P. Sorenson, "Testing Software Assets of Framework-Based Product Families during Application Engineering Stage," *Journal of Software*, Vol. 3, No. 5, 2008, pp. 11-25.

[35] R. Kauppinen, J. Taina and A. Tevanlinna, "Hook and Template Coverage Criteria for Testing Framework-Based Software Product Families," *Proceedings of the International Workshop on Software Product Line Testing*, Boston, 2004, pp. 7-12.

[36] B. Meyer, "Applying Design by Contract," *IEEE Computer*, Vol. 25, No. 10, 1992, pp. 40-51. doi:10.1109/2.161279

[37] G. Froehlich, "Hooks: An Aid to the Reuse of Object-Oriented Frameworks," Ph.D. Thesis, University of Alberta, Department of Computing Science, Edmonton 2002.

[38] N. Goel, A. K. Tripathi and M. Gupta, "Hook_Test: An Aid to the Hook-Driven Test-First Development of Framework Based Application," *Accepted for Publication in International Journal of Computer Science Issues*.