*J. Software Engineering & Applications*, 2010, 3: 455-459
doi:10.4236/jsea.2010.35051 Published Online May 2010 (http://www.SciRP.org/journal/jsea)

# Variability-Based Models for Testability Analysis of Frameworks

**Divya Ranjan[1], Anil Kumar Tripathi[2]**

[1]Dept. of Computer Science, Faculty of Science, Banaras Hindu University, Varanasi, India; [2]Dept. of Computer Engineering, Institute of Technology, Banaras Hindu University, Varanasi, India.
Email: ranjan_divya@yahoo.co.in, aktripathi.cse@itbhu.ac.in

Received March 20th, 2010; revised April 3rd, 2010; accepted April 5th, 2010.

## ABSTRACT

*Frameworks are developed to capture the recurring design practices in terms of skeletons of software subsystems/ systems. They are designed 'abstract' and 'incomplete' and are designed with predefined points of variability, known as hot spots, to be customized later at the time of framework reuse. Frameworks are reusable entities thus demand stricter and rigorous testing in comparison to one-time use application. It would be advisable to guaranty the production of high quality frameworks without incurring heavy costs for their rigorous testing. The overall cost of framework development may be reduced by designing frameworks with high testability. This paper aims at discussing various metric models for testability analysis of frameworks in an attempt to having quantitative data on testability to be used to plan and monitor framework testing activities so that the framework testing effort and hence the overall framework development effort may be brought down. The models considered herein particularly consider that frameworks are inherently abstract and variable in nature.*

***Keywords***: *Object-Oriented Frameworks*, *Variability*, *Customizability and Testability*

## 1. Introduction

Frameworks represent semi-codes for defining and implementing time-tested highly reusable architectural skeleton design experiences and hence become very useful in development of software applications and systems. As per Gamma *et al.* [1], famous in reuse literature as GoF, an object-oriented framework is a set of cooperating classes that make up a reusable design for a specific class of software which provides architectural guidance by partitioning the design into abstract classes and defining their responsibilities and collaborations. Being a reusable pre-implemented architecture, a framework is designed 'abstract' and 'incomplete' and is designed with predefined points of variability, known as ***hot spots***, to be customized later at the time of framework reuse [2]. A hot spot contains *default and empty interfaces*, known as ***hook methods***, to be implemented during customization [3,4]. Applications are built from frameworks by extending or customizing the framework, while retaining the original design. New code is attached to the framework through hook methods to alter the behavior of the framework. Hook descriptions provide guidance about how and where to perform the changes in the hook method to fulfill some requirement within the application being developed. With the help of *hook descriptions*, the framework developer passes his knowledge about what needs to be completed or extended in the framework, or what choices need to be made about parts of the framework in order to develop an application using the framework to framework users so that user is able to easily understand and use the hook. During framework reuse, the variant implementations of one or more hook methods, as needed, are created [5]. The code that the framework reuser writes, in order to create hook method implementations, is known as *application specific code* or *customized code.*

Frameworks are developed to capture the recurring design practices in terms of skeletons of software subsystems/ systems. It focuses mainly on the similarity amongst skeletons by identifying commonalities amongst them in terms of commonalities in the structure and functionality exercised by the concerned structure making use of the objects involved. It has been widely understood that the possibilities of variations provided by the hot spots and the corresponding hook methods etc. in the semi-code make it possible for a framework to be reused as extensively as permitted by the hot spots and the corresponding hook methods. The idea is simple that a framework permits variations across the application/ systems and attempts to design and code the common

 

parts and aspects of the structure.

One has to be very careful about developing *fault free reusable frameworks* because if the framework contains defects, the defects will be passed on to the applications developed from the framework [6]. The reusable framework thus demands stricter and rigorous testing in comparison to one-time use application [7,8]. It would be advisable to guaranty the production of high quality frameworks without incurring heavy costs for rigorous testing. This calls for analyzing testability of reusable artifacts so as to reduce the overall cost of framework based development.

Several techniques are specifically proposed to test object-oriented frameworks [1,6,9-12] and their instantiations [11,13-16].

There has been a little discussion upon the testability of frameworks in literature. As per Jeon *et al.* [2], the four factors that have direct influence upon framework testability are: controllability, sensitivity, observability and oracle availability. Ranjan and Tripathi [17] identified various factors and sub factors that affect the testability of frameworks so as to take care of those factors to bring high testability in frameworks. As per their observations, the factors that affect the testability of a framework are related to the characteristics of documentation of a framework, domain of a framework, design of a framework and the test support available for the framework testing like test tools, environments, reusable test artifacts and built-in tests etc. The concept of built-in tests is brought to frameworks with the intention of enhancing their testability [2,11,12,18].

In spite of wide importance and promotion of frameworks, over the last decades, a widely accepted set of measures to quantify its characteristics has not been established. Moreover, there is a complete lack of framework testability metrics related studies in literature that could produce quantitative data on testability to be used to plan and monitor framework testing activities so that the framework testing effort and hence the overall framework development effort may be brought down. This paper proposes framework testability models that consider that frameworks are inherently abstract and variable in nature.

The paper is organized in four sections. Few important reasons behind the need of framework testability analysis are presented in Section 2, the proposed testability models for software frameworks appear in Section 3 and finally Section 4 presents conclusions.

## 2. Need of Framework Testability Analysis

Some obvious reasons for rigorous testability analysis of a framework could be summarized as:

1) A testable framework ensures *low testing cost* and helps in reduction of overall development cost of a framework which has been designed and implemented as a semi-code.

2) Frameworks are reusable entities and hence high testability is essential. As a testable system is known to provide increased reliability [19,20].

3) High testability brings *high reusability*. Many a times a framework reuser will want to test few features to assess its quality. If testing a framework is tough then framework reuser will hesitate in testing and using the framework and will seek to choose another framework or go for development without deploying a framework.

4) To calm obvious scientific curiosity that while writing test cases for frameworks why it is tougher in some case than the other cases or, so to say, why for one framework we had to think very hard before we were able to write a meaningful test suite, whereas for other frameworks we could generate test cases in a straightforward way.

5) Testability holds a prominent place as part of the *maintainability* characteristic in ISO 9126 quality model ISO, 1991, so this study also increases our understanding of software quality in general [21].

6) Framework testability analysis creates a base for formulating the strategy for designing highly testable frameworks *i.e. framework design for test* (FDFT).

## 3. Testability Models Considering Abstract and Variable Nature of the Frameworks

This section aims at discussing various metric models for testability analysis of software frameworks that consider that frameworks are inherently abstract and variable in nature and thereby they provide opportunity to develop multiple applications with its reuse.

### 3.1 A Testability Model Considering the Diversity/Commonality of Applications that the Framework Represents

Frameworks represent a set of applications that share commonalities. During design of a framework, the common aspects are concretely defined and are known as *fixed spots* whereas the variable aspects are designed abstract and are known as *hot spots*. This may not always be easy to work out a framework definition in terms of the variable aspects that it is going to deal with. However a crude measure may suggest that as many would be the variable aspects that difficult its testing is likely to be. Thus,

$$TE_{Fr} = f(\textit{Variable Aspect Fraction}) \quad (1)$$

where,

$$\textit{Variable Aspect Fraction} \propto \frac{Aspects_{Var}}{Aspects_{Var} + Aspects_{Com}} \quad (2)$$

$Aspects_{Var}$ = Variable aspects in a framework;

$Aspects_{Com}$ = Common aspects in a framework;

By variable aspect fraction, we mean the ratio of vari-

 

able aspects with respect to common aspects in the framework.

This fraction may be calculated before the design of the framework to get an idea whether the candidate framework will be testable or not.

$$Tb_{Fr} \propto \frac{1}{Variable\ Aspect\ Fraction} \quad (3)$$

This model is a preliminary one which just gives the idea of testability of frameworks before proceeding for its design. The next models give the idea of testability after a framework is coded or at least designed.

### 3.2 A Testability Model Considering Variability in terms of Hook Methods Provided by the Frameworks

Only an idea can be formed about the testability of the candidate framework through the above model. It can better be judged once the design of the framework is ready in terms of hot spots and the constituent hook methods. Hot spots consist of hook methods which represent points of variability by providing the calling interface to variable tasks [22]. Variability is number of possible variant implementations of framework's abstract behaviors. The variability within the family of architectural skeletons is constituted into the hot spots of a framework [4]. It is variability that makes one instantiation of the framework different from others [22].

Variability of a framework may be determined by summing up the measures of variability of each hot spot in the framework. The variability of a hot spot depends upon the number of possible alternative implementations of each its constituent hook methods.

We, thus, can express variability of a framework as below:

$$VAR_{Fr} \propto \sum_{i=1}^{N_{Hotspots}} \left( \sum_{j=1}^{N_{HMi}} N_{IHMij} \right) \quad (4)$$

where,

$VAR_{Fr}$ = Variability of a framework;

$N_{Hotspots}$= Total number of hot spots in the framework;

$N_{HMi}$ = Total number of hook methods in $i^{th}$ hot spot;

$N_{IHMij}$ = Total number of possible implementations of $j^{th}$ hook method in $i^{th}$ hot spot;

A discussion that the variability how affects testability of frameworks appears in [17]. Testing a framework requires testing possible implementations [6]. Hence, more the variability of a framework more the testing effort is required. We can write,

$$TE_{Fr} \propto VAR_{Fr} \quad (5)$$

Therefore, combining (4) and (5), testability of a framework is:

$$Tb_{Fr} \propto \frac{1}{\sum_{i=1}^{N_{Hotspots}} \left( \sum_{j=1}^{N_{HMi}} N_{IHMij} \right)} \quad (6)$$

The testability of frameworks, thus, is inversely proportional to the total number of possible implementations of all hook methods in all hot spots. Alternatively, more the number of possible implementations of hook methods, more the effort is required for the testing of the framework.

Further, while calculating testing effort we find that certain variations require less effort in their implementations and thus incur lesser share in testing effort. The above model does not take this aspect into consideration and thus a stronger model is required. The next testability model which is based on the customizability of the framework is a stronger model than the present one, as it also considers the effort required for implementing variants of hook methods.

### 3.3 A Testability Model Considering Customizability of Hook Methods Provided by the Frameworks

Framework is customized by framework reusers to create concrete application software systems. The *customizability* of a framework may be interpreted by knowing how easy it is to customize (tailor) the framework. A framework is customized either by sub-classing (in case of white-box frameworks) or by composing preexisting components (in case of black-box frameworks). A testable framework should be highly customizable so that during testing of the framework various instantiations can be easily produced and subsequently tested. A discussion that the customizability how affects testability of frameworks appears in [17]. The customization of a hook method may require following:

**1) Changing** some object or method by the means of the mechanisms like inheritance, extensions, configuration, parameterization, template instantiation etc. [23]. What changes to make is defined in the *changes* section of a hook method description and

**2) Making assumptions** regarding other objects or methods and **understanding the assumptions** that other objects or methods have to make regarding this hook method. What assumptions are to make is defined in the *pre-condition constraints, post-condition constraints, uses* and *participants* sections of a hook method description.

Thus, we may express *Customization Effort* (CE) required for implementing a variant of a *hook method*, as below:

$$CE_{ImHM} \propto \sum_{i=1}^{N_{Changes}} Change\_load_i \times \sum_{i=1}^{N_{Assump}} Assumption\_load_i \quad (7)$$

 

**Table 1. Applicability/intention of the proposed framework testability metric models**

| S.No | Category | Framework Testability Metric Model | Applicability of the Model |
|---|---|---|---|
| 1. | Testability models considering abstract and variable nature of frameworks | Testability Model Considering the Diversity/Commonality of Applications that the Framework Represents | To have an idea about framework testability even ***before starting the design*** of framework. |
| 2. | -- do-- | Testability Model Considering Variability in terms of Hook Methods provided by the Frameworks | When framework ***variability*** is prominent during design and development. |
| 3. | -- do-- | Testability Model Considering Customizability of Hook Methods provided by the Frameworks | When framework ***customizability*** is prominent during design and development. |

where,

$CE_{\text{ImHM}}$ = Customization Effort required for implementing a variant of a hook method (*HM*);

$N_{Changes}$ = Number of changes to be made in some object or method during implementation of the hook method;

$Change\_load_i$ = Heaviness of $i^{\text{th}}$ change;

$N_{Assump}$ = Number of assumptions involved regarding objects and their interactions;

$Assumption\_load_i$ = Heaviness of $i^{\text{th}}$ assumption;

It is for sure that in Equation (7) $\sum_{i=1}^{N_{Changes}} Change\_load_i$ as well as $\sum_{i=1}^{N_{Assump}} Assumption\_load_i$ will never be zero because customizing a hook method would require at least one *change* and involve assumptions regarding at least one *participant*.

Further the *CE* of one hook method, which consists of customizing or implementing all its possible variants, may be defined as following

$$CE_{HM} \propto \sum_{i=1}^{N_{IHMi}} CE_{ImHMi} \tag{8}$$

where,

$CE_{HM}$ = Customization Effort (*CE*) for a hook method;

$N_{IHMi}$ = Total number of possible implementations for the hook method;

$CE_{ImHMi}$ = *CE* required for $i^{\text{th}}$ implementation of the hook method;

Now we will consider the *CE* of the framework itself. It would consist of the *CE* of all the hook methods of all the hot spots. This relation can be expressed as follows:

$$CE_{Fr} \propto \sum_{i=1}^{N_{Hotspots}} \left( \sum_{j=1}^{N_{HMi}} \left( \sum_{k=1}^{N_{IHMij}} CE_{ImHMijk} \right) \right) \tag{9}$$

where,

$CE_{ImHMijk}$ = Customization Effort for implementing $k^{\text{th}}$ variant of $j^{\text{th}}$ hook method of $i^{\text{th}}$ hot spot;

$N_{IHMij}$ = Total number of possible implementations of $j^{\text{th}}$ hook method in the $i^{\text{th}}$ hot spot of the framework;

$N_{HMi}$ = Number of hook methods in $i^{\text{th}}$ hot spot of the framework;

$N_{Hotspots}$ = Total number of hot spots in the framework;

Since, the testing effort of the framework depends upon the customization effort of the framework. So, we can get the framework testability model based on the customizability of framework, from Equation (9) as below:

$$Tb_{Fr} \propto \frac{1}{\sum_{i=1}^{N_{Hotspots}} \left( \sum_{j=1}^{N_{HMi}} \left( \sum_{k=1}^{N_{IHMij}} CE_{ImHMijk} \right) \right)} \tag{10}$$

Each of these testability metric models has different intention or applicability which is discussed in the table (**Table 1**) below, however, more than one model may also be employed at the same time.

## 4. Conclusions

It is mainly the *incomplete* and *abstract* nature of frameworks that makes it difficult to test than an object-oriented software system. In the present paper we proposed few preliminary framework testability models that consider that frameworks are inherently abstract and variable in nature. These models could produce quantitative data on testability to be used to plan and monitor framework testing activities so that the framework testing effort and hence the overall framework development effort may be brought down. Authors found a complete lack of framework testability metrics related studies in literature.

 

## REFERENCES


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

[2] T. Jeon, S. Lee and H. Seung, "Increasing the Testability of Object-oriented Frameworks with Built-in Tests," *Lecture Notes in Computer Science*, Vol. 2402, January 2002, pp. 873-881.

[3] G. Froehlich, H. J. Hoover, L. Liu and P. Sorenson, "Designing Object-oriented Frameworks," *CRC Handbook of Object Technology*, CRC Press, 1998, pp. (25)1-21.

[4] W. Pree, "Design Patterns for Object-oriented Software Development," Addison-Wesley, 1995.

[5] H. A. Schmidt, "Systematic Framework Design by Generalization," *Communications of the ACM*, Vol. 40, No. 10, October 1997, pp. 48-51.

[6] 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.

[7] J. S. Poulin and J. M. Caruso, "Determining the Value of a Corporate Reuse Program," *Proceedings of the IEEE Computer Society International Software Metrics Symposium*, Baltimore, May 1993, pp. 16-27.

[8] E. J. Weyuker, "Testing Component-based Software: a Cautionary Tale," *IEEE Software*, Vol. 15, No. 5, September 1998, pp. 54-59.

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

[10] M. E. Fayad and D. C. Schmidt, "Object-oriented Application Frameworks," *Communications of the ACM*, Vol. 40, No. 10, October 1997, pp. 32-38.

[11] 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, March 2000, pp. 7-12.

[12] 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, August 2002, pp. 25-34.

[13] 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, May 2008, pp. 595-604.

[14] 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, May 2005, pp. 169-196.

[15] 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, May 2008, pp. 11-25.

[16] 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.

[17] D. Ranjan and A. K. Tripathi, "Testability Analysis of Object-oriented Frameworks," *The Journal of Defense Software Engineering*.

[18] M. E. Fayad, Y. Wang and G. King, "Built-in test reuse," In: M. E. Fayad, *et al.*, Ed., *The Building Application Frameworks*, John Wiley and Sons, 1999, pp.488-491.

[19] R. V. Binder, "Design for Testability in Object-oriented Systems," *Communications of the ACM*, Vol. 37, No. 9, September 1994, pp. 87-101.

[20] J. M. Voas and K.W. Miller, "Software Testability: the New Verification," *IEEE Software*, Vol. 12, No. 3, May 1995, pp. 17-28.

[21] "Software Engineering-Product Quality," *ISO/IEC* 9126, 2001.

[22] G. Succi, A. Valerio, T. Vernazza, M. Fenaroli and P. Predonzani, "Framework Extraction with Domain Analysis," *ACM Computing Surveys*, Vol. 32, No. 1, March 2000.

[23] G. Butler, "Object-oriented Frameworks," 15*th European Conference on Object-Oriented Programming*, Tutorial Budapest, 2001.


 