Hook_Test: An Aid to the Hook-Driven Test-First Development of Framework based Application

Noopur Goel
Department of Computer Science,
Banaras Hindu University
Varanasi, Uttar Pradesh 221005,
India

A. K. Tripathi
Department of Computer Science and Engineering, Indian Institute of Technology,
Banaras Hindu University
Varanasi, Uttar Pradesh 221005,
India

Manjari Gupta
Department of Computer Science,
Banaras Hindu University
Varanasi, Uttar Pradesh 221005,
India

ABSTRACT
Enhanced quality with reduced cost and reduced time-to-market is the primary goal of any software industry. Researchers and practitioners are trying to aspire it with many techniques. Object-oriented framework is the promising technology to promote reuse, thus realizing desired goal. Inherently complex design and large size of a framework make it difficult to understand the framework, thus inhibit the purpose of reuse of framework. Conventionally, test is performed after the implementation phase of the waterfall model and any fault detection at this stage is a very costly affair. In this paper, we are introducing *Hook_Test* document to assist in test-first development approach of instantiation of framework known as *Hook-Driven Test-First Development* (HDTFD) of framework based application. Hook_Test guides the user of the framework to generate hook method specification based test cases for different types of hooks. These test cases can be further customized during the framework instantiation according to the user specific instantiation of the framework. Besides many advantages, the proposed approach for instantiation process of the framework is very simple and easy to understand. Hook_Test description and HDTFD approach are our contributions in this paper.

General Terms
Reuse-based Software Development

Keywords:
Framework instantiation, test-first development, Framework Interface Class (FIC), test cases, hook method specification.

1. INTRODUCTION
Quality assurance is highly demanded in any software industry, and is enhanced by performing software testing, 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 indispensible part of the software development process which itself is a highly time consuming and costly affair. The basic aim of testing is that a prevented bug is better than a detected and corrected bug [1]. Beizer [1] advocates that "first test, then code." The question to be pondered over is "What technique should be used to enhance the quality of the software while keeping the reduced cost and reduced time-to-market?" This question is of common interest for all researchers and practitioners in the field of software engineering aspire to achieve the goal. For a fast and efficient testing, testing process must be reuse-oriented and must be performed from the early stage of the lifecycle. In this respect, we are proposing an approach 'Hook-Driven Test-First Development **(**HDTFD)' for development of application based on framework. Assets reused in the HDTFD approach are framework and test cases developed from framework itself. Tests are introduced early in the requirement specifications phase in the form of test cases.

Frameworks are one of the promising reuse technologies. 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 [2]. 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 [3]. 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 [4].

During the instantiation phase, the application developer adapts the framework according to the application specific needs to create applications [5]. Understanding the "design" and "how to use" a framework needs a lot of learning time; framework instantiation is not very easy and inhibits the purpose of framework i.e. reuse.Proper documentation alleviates the steep learning curve fostering the practice of reuse.

Hook description [4] is used for many possible implementations of the **FrameworkInterface Classes** (**FIC**), shown in fig-1, for developing applications in the application development stage [6].

Reusable test cases can be built during the framework development stage to be used "as-is" or "customized" at application development stage to implement the FICs. Means for documenting and communicating test cases for each hook between framework builder, application developer, and tester is crucial.

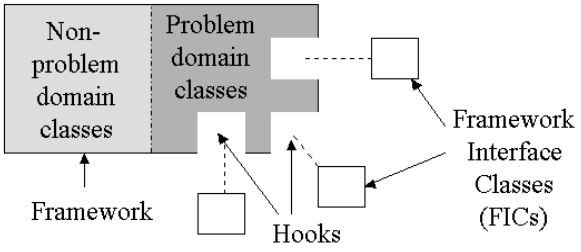

Fig 1: Framework Interface Classes (FIC) [6]

As per our knowledge, no work has been done till date on test description for the development of applications based on frameworks. We are proposing a novel technique- **Hook_Test** for documenting the test with respect to each hook. Hook_Testcan be used to generate the implementations according to the application specific needs. Hook method specifications are converted in the form of test cases, which are further reused, customized and adapted according to user specific needs during the framework instantiation. In our approach Hook_Test document would also be deployed along with the framework to the user of the framework.

Test-first development approach [7] is extended in the context of framework based application development to perform the test from the very beginning of the life-cycle. The basic idea behind test-first development (TFD) is write the test first, and then write the code to pass the test. HDTFD approach for the framework based applications aids in enhancing the quality of the application.

Hook_Test description assists to realize the instantiation of framework using the Hook-Driven Test-First Development approach.

The approach proposed in this paper contributes in the following ways:

- Techniques to specify test cases at the hooks of the framework.
- An approach to specify the instantiation process for different levels of hooks, which hold the constraints during the instantiation process of the framework, thus maintaining the consistency.
- Comparative study of various test cases for customization at different level of support.
- A case study in the domain of Drawing Editor is performed which describes the approach proposed and is implemented in Java using JUnit.

The proposed approach has the following advantages:

- Hook_Test description reduces the steep learning curve of the framework, thus reducing the time to market.
- Reduces the cost of the software by reusing the framework and reusing the test cases generated from the hook method specifications.
- Enhances the software quality by using contracts as test cases and applying the test-first development (TFD) process.
- Covers all the functionalities required for the specific application.

This paper is organized as follows. Section 2 provides an overview of related work on framework based application development. In section 3, the most common hook description is briefly discussed. The motivation behind the proposed work is discusses in Section 4. Section 5 describes theproposed approach *Hook-Driven Test-First Development* for instantiating the frameworks. Format of test description for each hook, *Hook_Test,*is proposed in section 6. Case study of the proposed approach is shown in section 7. Finally, section 8 concludes the paper.

2. RELATED WORK

Framework instantiation, the process of adapting the framework according to the application specific needs, is a complex task. Comprehending the detailed design and implementation of the framework under consideration is required before its instantiation. If documentation is not doneproperly, it takes a lot of learning time to understand the framework's design and how to adapt it, inhibiting the practice of reuse. Several methods of framework instantiation have been proposed by many researchers based on documentation of the framework- cookbook, pattern language, hook, examples of the applications derived from the framework and framework class hierarchy- providing the information to accomplish the required task. Cookbook is a type of guide, which assists the reuser of the framework to develop the needed application from the given framework through a set of related recipes. Framework instantiation performed by Krasner *et al.* [8] using the cookbook was much unstructured and much textual, leading to partially automate the instantiation process.Pree*et al*. and Sommerlad*et al.* [9][10], proposed the solution to the problem of interdependencies among the recipes by writing the recipes in hypertext and the active cookbook user interacts with the hypertext recipes through hyper-links. This approach automates the application development by stating certain features of framework adaption.

Contrasting Krasner *et al.* [8] method, Johnson [11] proposed another cookbook based instantiation method in which a set of patterns are structured as a directed graph and edges represent the references from one pattern to another. The first pattern describes the purpose of the framework and references to other patterns, and is the entry point to the directed graph. This approach does not provide solution to the problems occurring from the interrelation among the sub problems. This approach shows that patterns can also guide how to use the framework and fulfills the purpose of documentation [11] [12]. Braga *et al.* [13] have shown that pattern languages assist in instantiating the framework, sufficing the criteria that framework should be constructed on the same pattern language as described in [14]. Framework instantiation using the pattern languages comprise of four steps- system analysis, mapping between analysis model and framework, implementation of specific classes, and test of the resulting system.

Fontoura [5] and Lucena*et al.* [15] proposed an approach based on design patterns to document and instantiate the object-oriented framework, extending the standard UML. They improved standard UML to describe some features of design patterns by adding some annotations to highlight the hotspots and how to instantiate the frameworks. The drawback of this approach is that it does not elucidate what should be adapted in a framework to make user specific applications.

Ortigosa*et al.* [16] proposed a tool *HiFi (Helping in Framework Instantiation)* using the high level instantiation technique based on the functionalities provided by the given framework. The Intelligent Agent technology assists the framework user to select the required functionality for their application from the functionalities provided by the framework. Again, in assistance with the agent, a sequence of programming activities is carried out in order to implement it. Besides, the agent helps to execute the programming activities according to the design of the framework as documented in the rule based documentation approach proposed by the *SmartBook*[17]methods. The drawback of this approach is that it is very document dependent. If any functionality is not documented, the agent is not able to provide any assistance. The environment does not provide information about the type

of functionalities-mandatory, optional or alternative- and although the dependencies are represented through instantiation rules but is not explicitly presented to the framework user.

Olivieira*et al*.[18] proposed the declarative approach of framework instantiation based on software processes. First, the framework extension points, where the framework should be adapted, are represented in a systematic way in UML-FI (UML profile for framework instantiation) [19]. Next, for the framework instantiation representation, the sequence of tasks required to build the specific application using the frameworkis defined using RDL [20]. At last, the framework instantiation analysis is performed to validate the consistency of the framework instantiation task with the help of a set of constraints tool.

The high-level framework instantiation approach based on Feature Model proposed in [21] provides the simplified version of the framework's functional and technological characteristics Framework instantiation tool kit is provided to the user of the framework. First, the set of models is produced from which the application developer will opt for the features they want to include in their application. These models must be developed during the framework development. UML class model, which provides the framework's static view and the features model [22],which is produced during the domain analysis phase [23] at the start of the framework development process are the inputs to the process. Both the models are related through a trace mechanism-a UML dependency relationship. <<Trace>> relates the framework's characteristics represented by the features model with the framework's design elements which implements the extension points, thus establishes relationship between the functional and technological characteristics of the framework.

Generative approach based on domain specific languages for framework instantiation [24] [25][26] is also proposed. Cechticky*et al*. [27] proposed the generative approach to instantiate component based framework with OBS (on Board Software) *Instantiation Environment*. The designers configure and assemble the framework and then configuration actions are used to automatically generate the instantiation code.

Froehlich *et al.* [28] proposed the structured textual language to document the framework- purpose of the framework and how to instantiate framework. Each hook provides solution to a particular problem. Thus, the large problems are solved using many hooks. Our approach modifies the hook description proposed by Froehlich *et al.* to generate test cases from hook specifications and deploy the test cases to application developer using the framework. The application developer customizes and extends test cases to instantiate the framework.

3. BACKGROUND

Hooks define the purpose of the framework and how it is intended to be used. Hooks define the way to extend or customize incomplete or abstract parts of the framework to build the specific application [4] [29]. Froehlich [29] provides a special purpose language and grammar in which the hook description can be written. The hook description includes the implementation steps and the constraints (i.e., pre-conditions and post-conditions) to be followed to build the application and therefore FIC methods follow these constraints.

Each hook description is written in a specific format made up of several sections.

Name: a unique name, within the context of the framework that identifies the hook.**Requirement:** a textual description of the problem the hook is intended to help solve. The framework builder anticipates the requirements that an application will have and describes hooks for those requirements. **Type:** an ordered pair consisting of the method of adaption used and the amount of support provided for the problem within the framework. **Area:** the parts of the framework that are affected by the hook. **Uses:** the other hooks required to use this hook. The use of a single hook may not be enough to completely fulfill a requirement that has several aspects to it, so this section states the other hooks that are needed to help fulfill the requirement.**Participants:** the components that participate in the hook. These are both existing and new components. **Changes:** the main section of the hook that outlines the changes to the interfaces, associations, and control flow amongst the components given in the participants section. **Constraints:** limits imposed on the hook, or on the use of the hook. **Comments:** any additional description needed.

The hook is categorized along two axes:

- method of adaption viz. enable, disable, augment, modify and add, and
- level of support viz. option, supported pattern, open and evolutionary.

Option Hook: The implementation of option hook is provided with the framework. According to the application specificneeds, the required implementation is chosen. Framework acts like the black-box one [4].

Supported PatternHook: In supported pattern hook, feature is enabled, augmented, removed or replaced by keeping the same specifications as defined by the framework hooks. Introduction about creating new sub classes, method overriding or specialization, or filling in the parameters is written in the changes section of the supported pattern hook. A parameter may be a simple variable, method, or component defined as an option hook or, the method or class implementation depends on the framework user. The unique conditions imposed on parameters or methods and the consequences of using the supported pattern on rest of the frameworkis described in the*constraint* part [4].

Open Hook: Feature can be added, replaced, removed or augmented for **open hook**sometimes in an unexpected way. The requirement of the hook method may also be changed by creating new classes which are not subclasses, defining new operation on the classes, and code replacement or removal happens along with those performed in supported pattern hook. No support is provided for the changes section of the hook description. The *constraint* partdescribes the consequences of using the open hook, has on rest of the framework [4].

Evolutionary Hook: Framework reuser can break the design of the framework while instantiating the framework at the evolutionary hook [4].

4. MOTIVATION

Frameworks are designed to be reused to produce the needed applications with enhanced quality keeping the low cost and reduced time-to-market. Keeping this aim in mind, we were motivated for approach proposed in the paper. The purpose of choosing the elements in the approach is described below in following subsections 4.1 and 4.2.

4.1 Why Hook-Driven Development?

Hooks provide the means to extend or complete the framework, or what choices need to be made about parts of the framework in order to develop an application using the framework. Constraints are imposed on the changes or extensions performed in the framework to instantiate it. Thus, hooks help to ensure that the changes made are compatible with the design and implementation of the framework. The specifications (i.e. pre-conditions and post-conditions) of hook methods and invariants of the framework interface classes, i.e. contracts [30] form the basis of generating test cases for the implementation of hook methods.

4.2 Why Test-First Coding?

Test cases are the unambiguous representation of the requirements, which a program must satisfy. Writing the testcases initiates the process of converting the requirements to design and to implementing code [31]. "Test-first code is not a testing technique." Test-first code is an analysis technique which decides that what is within the scope or out of scope of the implementation, encouraging for explicit description of what conditions one anticipates before writing the code. Test-first code is a design technique. As the system grows, the design is simplified as well as one becomes assured about the correct functioning of the system [7].

Although the primary motto of TFD is design, the tests are also important. Automation of test suits is also possible in Test-first development approach [32]. To make testing easier, most test–first programmers use a good number of interfaces and abstract classes. This ultimately enhances the flexibility and reusability of software [32]. Frameworks are also the reusable skeletal architecture providing flexibility and reusability through interfaces and abstract classes.

Thus, besides many of the advantages of using the TFD [33] [34], the three reasons suffice for the development of framework based applications through *HDTFD* Approach.

- Hook specifications form the basis for the development of reusable test cases,
- All the functionalities and requirements needed by the application are fulfilled, and
- Test effort for applications may be reduced.

5. HOOK DRIVEN TEST-FIRST DEVELOPMENT- A NEW APPROACH

When application developers use the FICs to implement their applications, they deal with the specifications of FICs described by the hooks in three ways [6]:

- By using them as defined.
- By ignoring the specifications for the behaviors that are not needed in implementing application requirements.
- 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 [6]:

- By reusing them as-is.
- By ignoring or modifying some of the reusable test cases.
- By adding some more test cases or building new test cases from the scratch.

The HDTFD approach for option, supported pattern, and open hooksare proposed here. Evolutionary hook can change the structure of the framework itself, so it is not taken into consideration.

5.1 Proposed HDTFD Approach for Option Hook Implementation

In general, it does not require any effort in the implementation and testing of option hook. It is mentioned in section 4, implementation of option hook is provided within the framework. The method of adaption for the implemented hook is provided with the hook description, so HDTFD approach is required just for adapting FICs containing the option hooks as depicted in fig-2. The test cases $T_{M,}$ representing the specifications of the option hook method, are deployed to the users of the framework. The application developer reuses the test cases T_M as deployed by framework developer. The reuser of the framework carries out no customization of $T_{M.}$

5.2 Proposed HDTFD Approach for Supported Pattern Hook Implementation

The application specific code for supported pattern hook using HDTFD is implemented as depicted in fig-3.The test cases $T_{M,}$ representing the specifications of the supported pattern hook method, are deployed to the users of the framework. In case of supported pattern hook method, input is according to the test scenario- Boundary value test, Equivalence partitioning test etc. The execution condition of the test case T_M is represented by pre-conditions and FIC invariants while the expected result is represented by post-conditions and FIC invariants. The application developers customize the test cases T_M as per the method of adaption into T_{MA}. The pre-conditions or post-conditions of the test cases can be customized, but the FIC invariants should not be modified.

5.3 Proposed HDTFD Approach for Open Hook Implementation

The application specific code for open hook using HDTFD is performed as depicted in fig-4. The test cases $T_{F,}$ T_C and T_M, representing the requirement specifications of the framework, the FIC invariants and the specifications of open hook method respectively, are deployed to the users of the framework. In case of open hook method, input is according to the test scenario- Boundary value test, Equivalence partitioning test etc. The application developer customizes the test cases T_C and/or T_M as per the method of adaption into T_{CA} and/or T_{MA}. User of the framework is not able to modify T_F. $T_F$must be followed before and after the implementation of the open hook method.

Note: Verification of program at class invariant and framework constraint level is beyond the scope of this paper

5.4 Comparative Study for Customization of Different Test Cases at Different Level of Support

Comparative table of customization of various test cases, T_M, T_C and T_F, for different levels of hooks is shown in table 1. It also shows that flexibility to customize basic test cases, developed and deployed with the framework, by the

application developer increases from option hook to supported pattern hook to open hook.

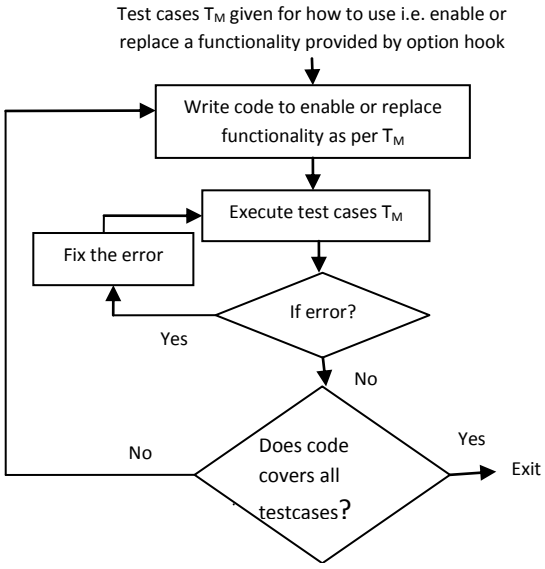

Fig 2: Proposed Option hook implementation using HDTFD approach

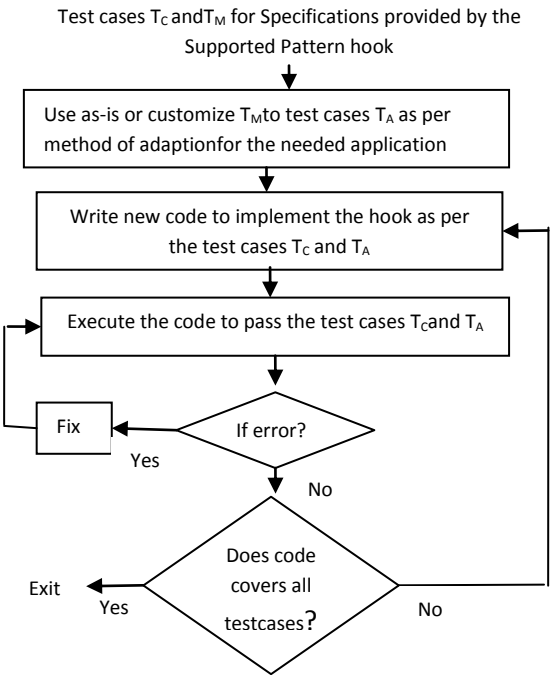

Fig3: Proposed Supported Pattern hook implementation using HDTFD approach

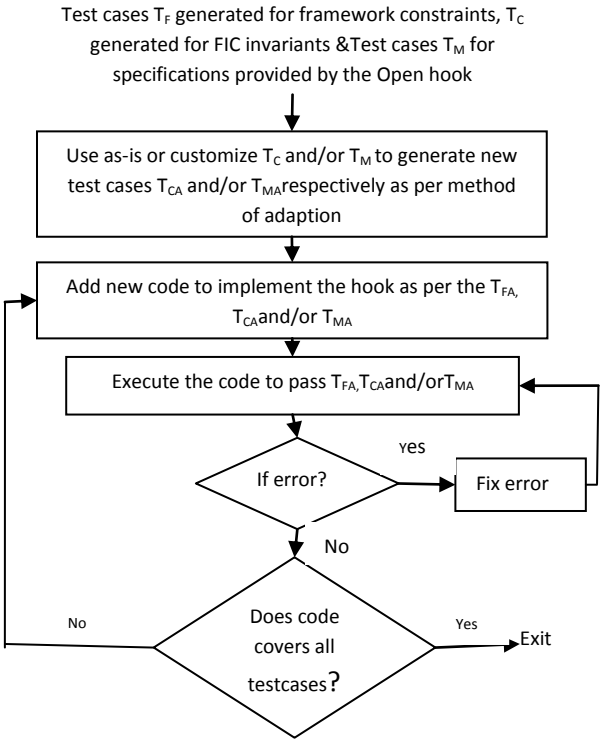

Fig 4: Proposed Open hook implementation using HDTFD approach

Table 1: Comparative study of customization of test cases T_M, T_C, T_F for hooks having different level of support.

Level of support / Method of Adaption	Option hook	Supported Pattern hook	Open hook
T_M	Mandatory	Optional	Optional
	Used as-is, i.e. $T_{MA}=T_M$	May be customized as T_{MA}	May be customized or generated from scratch as T_{MA}
T_C	Not required	Optional	Optional
	NA	Used as-is, i.e. $T_{CA}=T_C$	May be customized or generated from scratch i.e. T_{CA}
TF	Not required	Not required	Mandatory
	NA	NA	Used as-is, i.e. $T_{FA}=T_F$

Flexibility, to customize test cases, increases

Test cases deployed with the framework are as follows:

T_M : Test cases generated from hook method specifications.

T_C: Test cases generated from framework interface class invariants.

T_F : Test cases generated from framework constraints.

T_M, T_C, T_F are customized and adapted as test cases T_{MA}, $T_{CA,}$ T_{FA} respectively during framework instantiation.

6. HOOK_TEST: TEST DESCRIPTION WITH RESPECT TO EACH HOOK

The test cases generated with the help of the hook method specifications, known as basic test cases, support framework instantiation using HDTFD approach. To make the framework instantiation straightforward, need is felt to provide a new hook description- Hook_Testwhich modifies the previous one proposed by Froehlich*et al.*[4].

The Hook_Test description is complete in itself, such that the application developer can comprehend the requirements/functionalities supported by the hook in the form of test cases and support provided by the hook to customize basic test cases. Elements that form the test cases are input, execution condition and expected result. Each Hook_Test description consists of the following parts. Few of the sections that present basic information regarding hooks is same as proposed by Froehlich's hook description. Rests of the newly proposed subsections are explained in detail:

- **Name**: Unique name of the test description for each hook, corresponding to the name of hook in the hook description.
- **Task**: Requirements provided at hook, which it is intended to solve.
- **Type**: The ordered pair consisting of method of adaption used and level of support provided for test cases for the problem within the framework.Level of support may be *option*, *pattern* or *open test description* corresponding to the hook to be tested. It signifies that the basic test cases are tailored in line with the specifications of hook provided by the framework builder to be used as-is, extended or modified by the application developer. Method of adaption represents how the basic test cases, corresponding to functionalities provided by the hook, will be used: *as-is*, *augmenting to the basic test cases*, creating *new test cases from the scratch*, *ignoring some of the basic test cases or replacing the basic test cases*. Depending on the method of adaption and level of support provided by the framework builder, the application developer can customize the test cases aiding to automate the test script and generating the application specific code.
- **Area:** The parts of the framework that are affected by the hook.
- **Uses:** The other hooks required to use this hook. The use of a single hook may not be enough to completely fulfill a requirement that has several aspects to it, so this section states the other hooks that are needed to help fulfill the requirement.
- **Participants:** Components that participate in the hook. These are both existing and new components.
- **Changes:** The main section of the hook,which outlines the changes to the interfaces, associations, and control flow amongst the components given in the *Participants* section.
- **Test_id :**Identification of test case used to identify the test cases as per the combination with Test_Scenario and T_F, or T_C or T_M. Test_idis expressed as Test_01, Test_02, and so on.
- **Test_Scenario**: Input criteria used to generate basic test cases- functionality test, boundary value test, equivalence partitioning test, happy path test, negative test etc.
- **Test_cases**: Test cases T_M T_C and T_F, are generated from hook method specifications, FIC invariants and framework constraints respectively. For example, test case T_M is either pre-conditions of hook method or post-conditions of hook method or both. Test case T_C is FIC invariant and T_F is framework constraints.

  Before the execution of the hook method, the preconditions of the test case T_M must be satisfied for a given input. After the execution of the hook method, the post-conditions of the test case T_M must be satisfied. If $T_C$is provided with the framework, it must be satisfied before and after the execution of hook method. Similar is the case with T_F. Input depends on the Test_Scenario.

  In case of option hooks, none of the test cases except for *how to adapt* the hook is necessary, in case of supported pattern hooks, pre-conditions, post-conditions are optional while class invariants are mandatory and framework constraints are not needed. In case of open hooks, pre-conditions, post-conditions and framework interface class invariants are optional and framework constraints are mandatory.
- **Comments**: Any additional description, if needed.

All sections of the Hook_Test are not required for all hooks. All test cases T_F, T_C and T_M are also not required for all hooks. Those sections or test cases which are not required are simply left out.

7. CASE STUDY

In this section we are using Drawing Editor domain to describe our approach. We used Java for its implementation and test cases are generated using JUnit 3.0. Netbeansis

usedas an editor. The *Drawing* framework has an abstract class *Item* consisting ofa supported pattern hookmethod *Draw*. Fig-5 is the hook description of the *Draw* hook (as per the concept proposed by Froehlich *et al.* [4]) belonging to FIC *item.*

Name: Draw hook
Requirement: A new type of figure can be added to the drawing framework.
Type: Supported Pattern.
Area: Tools
Participants: BasePanel (Framework class)
Uses: None
Changes: 1. Obtain a reference to the graphics context of the container (**BasePanel**) using the getGraphics() method of the JPanel class in java.
2. The subclass must extend the I**tems** class and must not be abstract.
3. Draw the required figure using the appropriate drawing methods supplied by the draw method.
Pre-condition: 1. Drawing surface cannot be null i.e. **BasePanel** != null.
2. (x1, y1, x2, y2) are within the range of the drawing surface **BasePanel**.
3. (x1, y1) >= (0,0) and (x2, y2) >= (x1, y1)).

Fig 5: Hook description for *Draw* hook.

Fig-6 shows the *Draw_Test* corresponding to *Draw* hook in proposed Hook_Test format.

Name : Draw_Test
Requirement : A new type of figure can be added to the drawing framework.
Type : Supported Pattern.
Area : Tools
Participants :Basepanel (Framework class)
Uses : None
Changes: 1. Obtain a reference to the graphics context of the container (**BasePanel**) using the getGraphics() method of the JPanel class in java.
2. The subclass must extend the I**tems** class and must not be abstract.
3. Draw the required figure using the appropriate drawing methods supplied by the draw method.
Test_cases, T_M :
Pre-condition:
BasePanel != null.
x1=0, x2=0, y1=0, y2=0.
(x2>=x1 && y2>=y1).
(x2<=d.width&& y2<=d.height);

Fig 6: Hook _Test description for *Draw* hook, *Draw_Test.*

The Hook-Test description *Draw_Test*is deployed with the *Drawing* framework to the application developer.

Further, application developer uses the *Draw_Test* description to generate test case method *testDraw()*.Using the *Draw_Test* description test cases are written in *testDraw()*method as shown in fig-7.

```
public void testDraw() {
System.out.println("draw");
BasePanelbp = null;
int x1 = 0;
int y1 = 0;
int x2 = 0;
int y2 = 0;
assertTrue(x2>=x1 && y2>=y1);
bp=new BasePanel();
BaseFrame f=new BaseFrame();
      f.add(bp);
f.setVisible(true);
assertTrue(bp!=null);
      Dimension d=bp.getSize();
assertTrue(x1>=0 && y1>=0 && x2<=d.width&&
y2<=d.height);
      Circle instance = new Circle();
instance.draw(bp, x1, y1, x2, y2);
   }
}
```

Fig 7: Test case *testDraw* for draw hook

The *Changes* section of the *Draw_Test* description and the test case method *testDraw* aids in implementing *Draw* hook. Implementation of Draw hook is shown in fig-8.

```
public class Circle extends Items{
   @Override
public void draw(BasePanelbp, int x1, int y1, int x2, int y2) {
          Graphics g=bp.getGraphics();
int l=Math.min(x2-x1,y2-y1);
g.drawOval(x1, y1, l, l);
   }
}
```

Fig 8: Implementation of Draw hook method corresponding to test case *testDraw*.

8. CONCLUSION

The proposed work introduces a new approach to instantiate the object-oriented framework as per the functionalities needed by the user. Today's needs of the industries- enhanced quality with reduced time-to-market and reduced cost- urges for the practice of reuse. Object-oriented framework is the promising reuse technology catering the needs of the software industry. Understanding the "how to use" a framework improves the practice of reuse of the framework.

Conventionally, test is performed after the implementation phase of the waterfall model and any fault detection at this stage is a very costly affair. In this paper, we have introduced a hook based test description, *Hook_Test*, to assist in the test-first development of software based on object-oriented framework. The hook based test description Hook_Testis delivered along with the framework to the user for instantiating framework as per the specific needs. Reuser of the framework can very easily know how to instantiate the framework under the guidance of test cases and changes subsection provided by the Hook_Test description. Besides many subsections, the Hook_Test description consists of the method specifications, framework interface class invariants and framework constraints which are used to generate test cases. These test cases further form the basis for *Hook-Driven Test-First Development* of specific framework instantiation.

We described our proposed approach in this paper by an example in the domain of Drawing Editor. In our future work,

we are studying the testability of the framework based application developed using HDTFD approach.

9. REFERENCES

[1] B. Beizer, "Software testing techniques",2nd edition, Dreamtech press.

[2] K. Beck and R. Johnson, "Patterns Generate Architecture", in Proceedings of 8th European Conference on Object Oriented Programming, Bologna, Italy, 1994, pp. 139-149.

[3] 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.

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

[5] M. F. M. C. Fontoura, "A Systematic Approach to Framework Development", Ph.D. Thesis, Computer Science Department, Pontifical Catholic University of Rio de Janeiro, Rio de Janeiro, Brazil. 1999. Available from: http://www.teccomm.les.inf.pucrio.br/publicacoes.htm [Accessed 5 May 2012].

[6] J. A. Dallal, "Class-based Testing of Object-oriented Framework Interface Classes", Ph.D. Thesis, Department of Computing Science, University of Alberta, City, Country, 2003. Received through email: {j.aldallal@ku.edu.kw} date: 21/05/2012.

[7] K. Beck, "Aim, Fire [test-first coding]",IEEE SOFTWARE, Vol. 18, No. 5, September / October, 2001, pp. 87-89.

[8] G. E. Krasner and S. T. Pope, "A Cookbook For Using The Mode-View-Controller User Interface Paradigm in the Smalltalk-80", Journal of Object-Oriented Programming, Vol. 1, No. 3, 1988, pp. 26–49.

[9] W. Pree, G. Pomberger, A. Schappert and P. Sommerlad, "Active Guidance of Framework Development", "Software-Concepts and Tools", Vol. 16, No. 3, 1995, pp. 136–145.

[10] P. Sommerlad, A. Schappert and W. Pree, "Automated Support for Development with Frameworks", in Proceedings of the 17th International Conference on Software Engineering on Symposium on Software Reusability, ACM Press, New York, 1995, Vol. 20, pp. 123–127.

[11] R. E. Johnson, "Documenting Frameworks using Patterns", in ACM OOPSLA, 1992, Vol. x, pp. 63-76.

[12] D. Brugali and G. Menga, "Frameworks and Pattern Languages: An Intriguing Relationship", "ACM Computing Surveys", Vol. 32, No. 1, 1999, pp. 2–7.

[13] R. T. V. Braga andP. C. Masiero, "The Role of Pattern Languages in theInstantiation of Object-Oriented Frameworks", in OOIS 2002 Workshops, LNCS 2426, _Springer-Verlag Berlin Heidelberg 2002, pp. 122–131.

[14] R. T. V. Braga and P. C. Masiero, "A process for framework construction based on a pattern language", in Proceedings of the 26th Annual International Computer Software and Applications Conference, IEEE computer Society, Oxford-England, 2002, pp. 615-620.

[15] M.F.M.C. Fontoura and C.J.P. Lucena, "Extending UML to improve the representation of design patterns", Journal of Object-Oriented Programming, Vol. 13, No. 11, 2001, pp.12–19.

[16] A. Ortigosa, M. Campo and R. Moriyon, "Towards agent-oriented assistance for framework instantiation", in OOPSLA '00 Proceedings of the 15th ACM SIGPLAN conference, ACM Press, New York, 2000, Vol. 35, pp. 253–263.

[17] A. Ortigosa and M. Campo, "SmartBooks: A Step Beyond Active-Cookbooks to aid in Framework Instantiation", "Technology of Object-Oriented Languages and Systems" 25, IEEE Press, 1999, pp. 131-140.

[18] T. C. Oliveira, S. Paulo, C. Alencar, I. M. Filho, Carlos J.P. de Lucena, and D. D. Cowan, "Software Process Representation and Analysis for Framework Instantiation", IEEE Transactions On Software Engineering, Vol. 30, No. 3, March 2004,Pp. 145-159

[19] T. C. Oliveira, P. S.,C. Alencar, and D. D. Cowan, "Towards a Declarative Approach to Framework Instantiation", In Proceeding of First Workshop Declarative Meta-Programming (DMP), 2002, pp. 5-9.

[20] C. T. Oliveira, P. S. C.Alencar, C. J.P. de Lucena and D. D. Cowan, "RDL: A Language for Framework Instantiation Representation", The Journal of Systems and Software, Vol. 80, No. 11, 2007, pp. 1902–1929.

[21] I. M. Filho, T. C. Oliveira and C. J. P. de Lucena, "A Framework Instantiation Approach Based on the Features Model", The Journal of Systems and Software, Vol. 73, No. 2, 2004, pp. 333–349.

[22] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak and A. S. Peterson, "Feature-Oriented Domain Analysis (FODA) Feasibility Study", Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania. 1993.

[23] K. C. Kang, S. Kim, J. Lee, K. Kim, E. Shin and M. Huh, "FORM: A Feature-Oriented Reuse Method with Domain-Specific Reference Architecture", Annals of Software Engineering, Kluwer Academic Publishers, Dordrecht, Holland, Vol. 5. No. 1, 1998, pp. 143–168.

[24] K. Czarnecki, T. Bednasch,, P. Unger, and U. Eisenecker, "Generative Programming for Embedded Software: An Industrial Experience Report", in Proceedings of the 23rd Conference on Generative Programming and Component Engineering, LNCS Springer-Verlag 2002, Vol. 2487, pp. 156-172.

[25] K. Czarnecki and U. Eisenecker, "Components and Generative Programming", in Proceedings of the Joint 7th European Software Engineering Conference and ACM SIGSOFT International Symposium on the Foundations of Software Engineering, France, 1999, Vol. 24, No. 6 pp. 2-19.

[26] G. Butler, "Generative Techniques for Product Lines", Software Engineering Notes, Vol. 26, No. 6, 2001, pp. 74-76.

[27] A. Pasetti, W. Schaufelberger, F. Pfenning and Y. Smaragdakis, "A Generative Approach to Framework Instantiation", Vaclav Cechticky1, Philippe Chevalley2, (Eds.): GPCE 2003, LNCS 2830,.© Springer-Verlag Berlin Heidelberg 2003, pp. 267–286.

[28] G. Froehlich, H.J. Hoover, L. Liu, P. Sorenson, "Hooking into object-oriented application frameworks", in Proceedings of the 19th International Congress on Software Engineering, ACM Press, New York, 1997, pp. 491–501.

[29] G. Froehlich, "Hooks: an aid to the reuse of object-oriented frameworks", Ph.D. Thesis, Department ofComputing Science, University of Alberta, 2002.

[30] Meyer, "Applying Design by Contract", IEEE Computer, Volume 25, No. 10, 1992, pp. x.

[31] S. Fraser, D. Astels, K. Beck, B. Boehm, J. McGregor, J. Newkirk and C Poole, Panel, "Discipline and Practices of TDD:(Test Driven Development)", in Proceeding OOPSLA '03 Companion of the 18th annual ACM SIGPLAN, 2003, pp. 268-270.

[32] D. S. Janzen and H. Saiedian, "Does Test-Driven Development ReallyImprove SoftwareDesign Quality?", IEEE Software, Vol. 25, No. 2, March/April 2008, pp. 77-84.

[33] T. Bhat and N. Nagappan, "Evaluating the Efficacy of Test-Driven Development: Industrial Case Studies", in ISESE'06 Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering, 2006, pp. 21–22

[34] D. S. Janzen,"On the Influence of Test-Driven Development on Software Design", in Proceedings of the 19th Conference on Software Engineering Education & Training, IEEE, 2006, pp. 141-148.