

Components Testing Using UML Use-Case & Scenario Diagram

¹Ms Shivani Jain, ²Mrs. Anju Saha,

¹M.Tech Fellow (Gate scholar), University School of Information Technology,
GGs Indraprastha University, Kashmere Gate, Delhi, shivanijain131@yahoo.co.in
²Lecturer, University School of Information Technology,
GGs Indraprastha University, Kashmere Gate, Delhi, anjukocchar@yahoo.com

Abstract: *With the increasing complexity of software system, coping with the same has become more complex for any human being. In response to the software crisis software development methodologies are developed to improve the quality of the software. Improvement can be realized through the thorough analysis and hard testing methodology. In recent years more and more developers have started relaying on components based system. Without appropriate knowledge of interface, it is always difficult to test the Component Based System. UML, a modeling language having different views of a system, is widely used for component testing. In this paper we test the components using use-case and scenario (behavior), and test suits are generated in Java language. In the Object Oriented Software Development methodology the expected functionality of a system is often identified as scenario, and these scenarios will be implemented by a set of interacting object. We shall call this approach CTU-SAUC (Component Testing Using Scenario And Use Case).*

This paper talks about a new method, which uses use-case and scenario for the components testing and enhances it further with activity diagram for the components based software testing. This will support testers in systematically developing test case generation for the components and components based software system.

Keywords Used

Components, Scenario, Use-case, Activity diagram, Assertion, Behavior.

Introduction

Allen had predicated that upto 70% of all new software intensive system will heavily rely on components based software. By using components, software quality is increased as well

as time is saved. Components are written in different platforms and languages, which are supposed to be platform-independent. It has been observed that though they work properly when deployed on a same platform, they behave differently when they are implemented on different platforms. Therefore adequate integration of re-useable component is the key to success in the component software industry. Usually black box testing is used for checking the functionality of the components/systems. On the other hand white box testing checks the quality of program that are used in components. Because of component heterogeneity and implementation transparency both the traditional methodology fail.

In this case our only hope is UML, a modeling language in which testing is done through the behavior of the system and test cases can be generated through the analysis diagrams.

Validation and verification are important activities in developing a software system. It is the process of determining whether the requirement for a component/system is complete and correct [3].

The production of each development phase fulfills the requirements or condition imposed by the previous phase and the final component/system compare the same with the specified requirement.

Organization of this paper

1. About the components
2. Use case and scenario
3. Why UML is best for component testing
4. Describe the assertion on the scenario
5. Methodology for the component testing
6. Practical example for checking the components

Components

Let us first begin with the basics. A component has been defined as a unit of composition with contractually specified interface and explicit context dependencies. A software component can be displayed independently and is subject to composition by the third party [2]. Now to interact with the components we need interfaces. Interfaces are the access points of components. Over and above we need events that are used for triggering an interface. Events can be defined as an incident. These may be triggered, through an exception and some action is performed. For e.g. in AWT classes, in Java, when button is pressed a specific action is performed.

The key to success of a reliable software system is ensuring the accuracy of interaction among the components. Interfaces are the most common way to activate components so it is necessary during integration and in system testing that we test each interfaces in the integrated environment at least once.

To observe possible behavior of each interface during run time, every interaction among the interface needs to be tested. Some events that are not triggered via interfaces may also have an impact on the components therefore they should also be checked and tested.

At this point we also need to discuss context-sensitive dependency relationship [1]. This not only includes direct interaction amongst interfaces and events but also indirect collaborative relationships. These relationships are indirect relationship caused by interaction of different layers of interfaces and events. Testing the context-sensitive dependency relationship may help us in identifying interoperability faults caused by improper interaction among different components. We can obtain the specification of an interface and a component by using UML activity diagram. This gives the control flow between different activities and can also help us in identifying the possible ways in which they can flow.

Activity diagram are used for showing the behavior of interface & components and use case and scenario used for the components testing.

Use case and Scenario

Use case diagram in UML are used for the requirement analysis. In use case we identify the actors and the basic functionality of the system. And show some of the relationship among the use cases.

Timing, data recourse and casual dependencies are not included in use case model. Other dependencies for complex structures are not encouraged because they support functional decomposition, which would describe functional structure than object-oriented structure.

Scenarios are used in many modern software engineering methods for capturing the requirement and specification of the system in wider terms. *Scenario is identified as a sequence of interaction among the components of the system for performing a particular task.* Such a scenario is typically implemented by a sequence of interaction among various objects.

Analysis scenarios are hardly ever used in testing. Only some researchers have worked on it. Scenarios form a kind of abstract test cases.

Testing is often done in unsystematic and unstructured way. Testing is hardly planned, documented and testing is not started until implementation has almost started.

When the system is developed the specification is clear to almost everyone and it is easy to write these specifications in natural language. However, natural language is ambiguous and vague. Here the concept of scenario comes handy. It would be desirable to have a language that is easy to use as natural language and as rigid, consistent and concise as formal language [1]. By using the scenario we try to walk between both, we use the ease of natural language and formalize it into scenario. We are then alleviated from some of the problem that we face using natural language.

Scenario, “an ordered set” of interaction between partners, usually between the system and set of actors external to the system, may comprise of a sequence of interaction steps or a set of possible interaction steps.

Scenario based approach

The scenario-based approach includes the following steps.

- Use of the natural language not only defines the requirement but also to validates the system.
- Uncovers contradictions, ambiguity and vagueness in natural language description to be formalized and also prioritize the requirement.
- For the pre/post conditions, data ranges, data values and non –functional requirement like performance are also to be considered, as it is very important issue in case of components development.

UML is best to describe the behavior of the components/system

- UML provides high level of information that characterizes the internal behavior of components, which can be processed efficiently and used effectively when tested.
- UML has emerged as the industry standard for software modeling notation and various diagram are available for define different prospects and angels of software.
- UML provides a set of models that can have different levels of capacity and accuracy for components modeling, that satisfy the real world need for the software.

These are ideal condition for testing the components and component based software.

Assertion on the scenario

Traditional software testers concentrates on testing the program when source code is available, while the approach described in this paper emphasis on the generation of test plan during the design phase. It collects test coverage information and writes the scenario modeled by the behavior contracts. A test is considered passed if none of these assertions are violated. The effectiveness of this technique depends on what properties will be asserted and when the assertion will be asserted and when the assertion will be evaluated.

These assertions [12] document the assumptions that the developers make at the boundaries of different modules. In this case we need to track the progress of the object interaction for each specific scenario. We need to check if

- The right objects have been participating in the scenario
- The actions performed by these objects at various steps are in the right order
- These actions have the expected effect on the program states under the context of this scenario.

A behavior contract specifies how various scenarios are expected to progress for performing the particular task during the program execution. Behavior contracts is very important in case of components as it specify how the components behaves with the outer world and it specify what are pre/post condition in this process. It specifies how object should behave and interact with one another under the context of scenarios in order to correctly perform a designated task.

Methodology for the component testing

In CTU-SAUC we propose a procedure to fully exploit the advantage of scenario in testing the components. We also show the dependencies among the scenarios for better description of components and how different interfaces interact with the system. For showing the context – dependency we use the dependency among the scenario. This enhanced method enables the tester in testing and thus supports improved test case development.

Scenario creation is an iterative process for the better understanding of components. Through the activity diagram we show the control flow of the components.

Scenario Creation Procedure

1. Find all Actors (internal/external)
2. Find all relative system, external events (in components testing identification is must)
3. Define input/output of the system in the ideal condition (possible paths)
4. Define system boundaries

5. Prioritize the boundaries, as the entire checking is not possible in any software
6. Create a step-by-step description of events and action of each scenario
7. Define the assertion on each step
8. Create an overall diagram and an activity diagram
9. Check the requirement specified.
10. Extend scenario by refining the scenario description; break down tasks to single working step.
11. Also identify the alternative flow of actions; specify exception and how to react to exceptions
12. Include the non-functional requirements
13. Scenario review by user and client

Dependency among the scenarios

Scenario dependency can be of different types like:

- Abstraction: behavior, inheritance, aggregation
- Temporal: sequence dependency
- Causal: depend on some condition may appear in alternative flow

Scenario can be implemented in sequence, parallel/concurrent and depending upon the interface.

Practical example for the testing of components

Testing should be performed on all the components. It should also be performed when integration takes place with the third party. It is observed that, if the behavior of components is well understood in early phase of development chances of error can be reduced upto 30-40% in any system. Let us take an example and understand the process.

Login, is an essential part of any software system. It is a similar procedure in every system but the constraint can be different depending upon the system requirement. A system can be a banking system, airline reservation system on which thousand of user are accessing the information or a library system on which hundreds of users are lending/returning books, where this login module is an integral part.

For this Login use case, components can be design, which has the same functionality in all systems mentioned above, but working on different platforms and different client systems. By defining the contracts on these properties we can easily define the pre/post condition for the Login components.

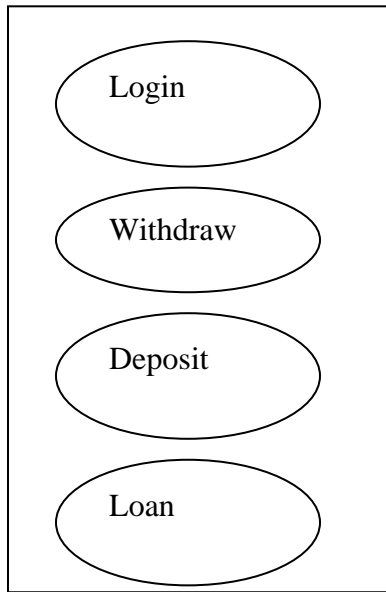
Next comes the Actor concept for each system. Every system has different actors. For example in a library system student and library in-charge are the main actors and for banking/airline system customer and database administrator are the main actors. Each of them has different functionalities and behavior according to their goal. So depending upon which user is going to operate the system, different contract can be applied at the analysis and design phase and reused in testing phase.

Different blocks contain different java statements that can access monitoring variables and access program state through same mechanism. Following procedure can perform computation necessary for testing

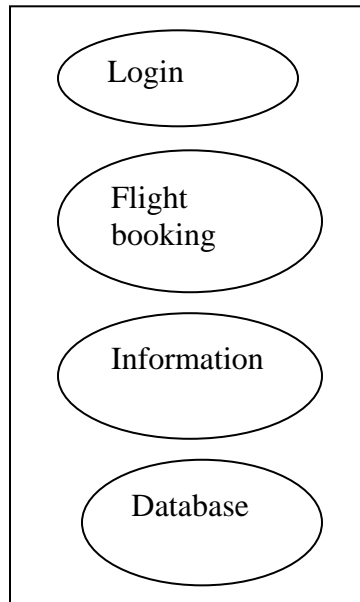
- Insert at node
- Insert key (password),
- Accepted or
- Failed, depending on the constraint apply on it.

Different systems showing Login module is a must so we choose login as a component development.

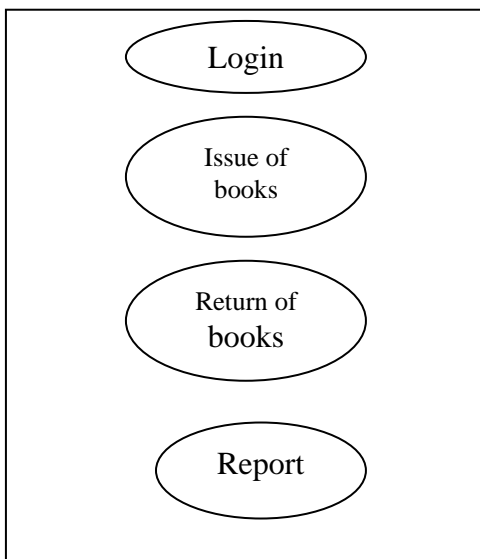
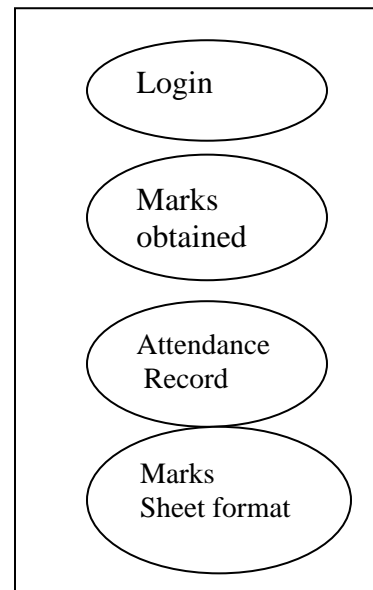
BANKING SYSTEM



AIR LINE SYSTEM



RESULT SYSTEM



For this system different actors can be:

- Student
- Faculty
- Administrator
- Library in charge
- Database administrator

Each of different actors has different behavior that gives different functionality to the system. So depending upon which user is operating the system, different parameters are assigned and then scenarios are generated. Like

Student login

1. Issue of books
2. Return of books
3. Enquires about the books
4. Information about the late fees

Faculty Login

1. Issue of books
2. Return of books
3. Enquires about the books
4. Issue of Generals, Research papers, Magazines, CD's.
5. Late Fee, if any

Administrator Login

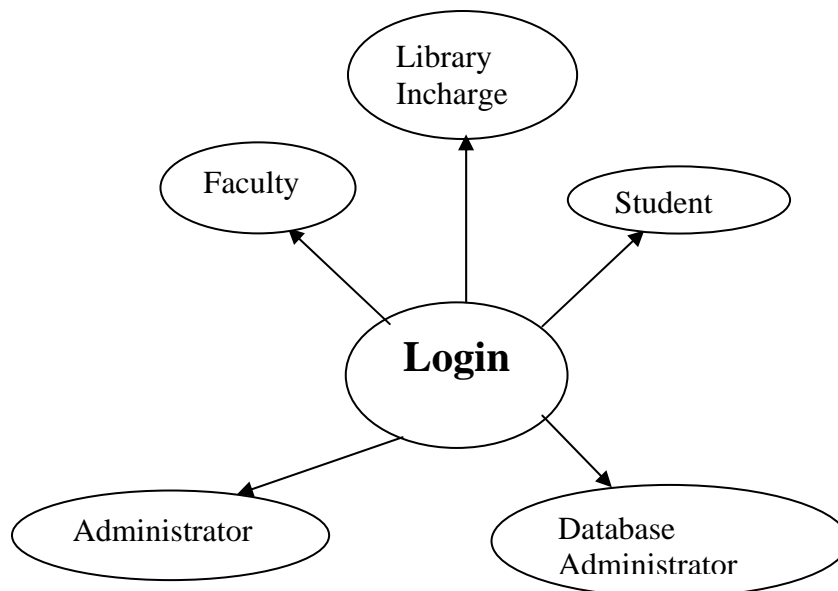
1. Enquiry about books available
2. Report analysis

Database Administrator

1. Entry of books, Students, Faculty
2. Modification in the same like insertion, deletion, Update the data

Library In charge

1. Report generation Daily, Weekly, Monthly, and Yearly
2. Complete Information about the books
3. Place the order for new books
4. Control the whole System.

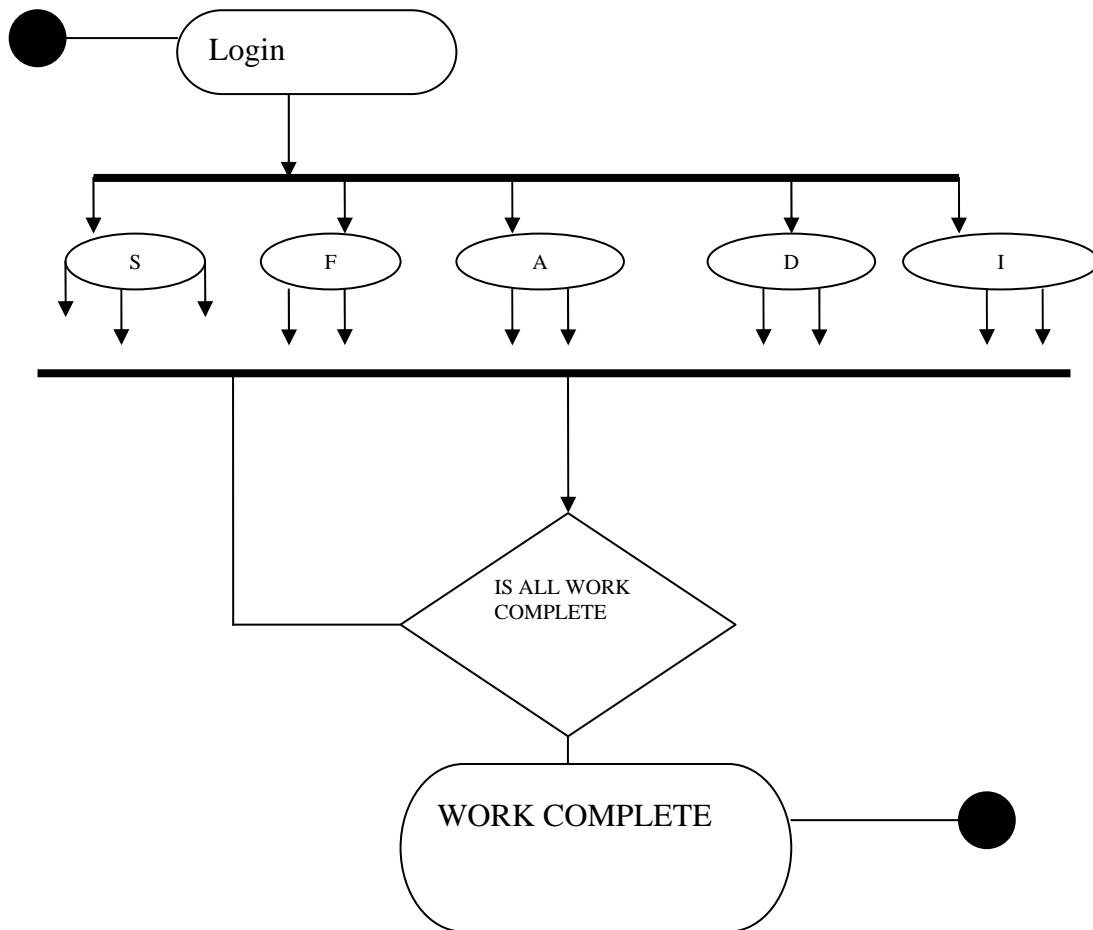


For each case different objects are identified for respective users and the developer defines the contracts on it after checking the user authentication.

library system. More than one user like student, faculty, administrator, library in-charges and database administrator can log in simultaneously. Once they have “logged-in” the system, their work is executed concurrently. After completion of their individual jobs the system verifies if the work is completed.

Activity Diagram for Control Flow Analysis

We now make the activity diagram for depicting the control flow among the different users of the



Conclusion and further work

In this paper we describe the methodology for the components testing using Use case and Scenario and then defining the assertion at the

different behavior level. Depending upon this behavior different path can be reorganized and these paths can be formalized through the Activity Diagrams. First, identify the components and their behavior, which is similar

for all different system working on platforms. In this way we can easily identify the objects, which are interacting with other systems and implement those objects in java language.

Right now we are working on the implementation phase for automatic generation of test case after identifying the objects in the components.

References used

[1] Riser, Jahannes and Glinz, Martin. *Management of Inter-Scenario Relationship Using dependency charts to Improve Scenario-Based Testing* Conference on Testing Computer Software TCS, 2000, Washington D.C., 2000.

[2] Wy, Ye Chen, Mei Hwa And Offutt, Jeff *UML – Based Integration Testing For Components –Based Software.*

[3] Kim, Youngchul And Carlson C.Robert *Scenario Based Integration Testing For Object Oriented Software Development.*

[4]Jacobson, I. Booch, G. And Rumbaugh, J. *The Unified Software Development Process.*

[5]Jacobcon, I. *Formalizing Use Case Modeling* Journal Of Object Oriented Programming Vol 8# 3, 1995.

[6] Ryser. J *A Practical Approach To Validate And Testing Software System Using Scenario*1999.

[7] Jacobson, I., Et Al, *Objet-Oriented Software Engineering: A Use Case Driven Approach*, Addison-Wesley/ACM Press, 1992.

[8] Binder, R. *Testing Object-Oriented Systems. Models, Patterns, And Tools* Addison-Wesley, 1999.

[9] Glinz, M. And Òan *Integrated Formal Method Of Scenarios Based On Statecharts*, European Software Engineering Conference, 1995.

[10] IEEE, *Standard Glossary Of Software Engineering Terminology* IEEE Std 610.12-1990: IEEE Computer Society Press, 1990.

[11] Jacobson, I. And Christerson, M. Growing, ÒA *Consensus On Use Cases*, *Journal Of Object-Oriented Programming*, Vol. 8, # 1, Pp. 15-19, 1995.

[12] Liang, Donglin. Xu, Kai. *Testing Scenario Implementation With Behaviour Contracts* COMPSAC 2006,IEEE, 2006.