International Journal of Engineering and Advanced Technology (IJEAT)
ISSN: 2249-8958 (Online), Volume-2 Issue-6, August 2013

Software Testing and Its Dependence on Software Architecture

Bibhuprasad Sahu

Abstract: ***The complexity of the soft ware's nowadays has become a central design problem. A system's architecture provides a model of the system that suppresses implementation detail, allowing the architect to concentrate on the analysis and decisions that are most crucial to structuring the system to satisfy its requirements. This paper defines a formal technique to test software systems at the architectural level using software Architecture Description Languages (ADL).ADLs uses testable components in the architecture. The use of independent and reusable components and their inter communication issues is very much useful in designing a flexible software. Formalized software architecture description languages provide a significant opportunity for testing because they precisely describe how the software should behave in high level view, and they can be used by automated tools. The basic theme in this paper is that the software designed using a formal approach (ADLs) can enable architecture based testing which in turn will lead to a robust software design. Software architectures, particularly when defined formally using some sort of architectural description language, can provide a description of the software system that could be used for test case generation at the system level. This enables developers to abstract away the unnecessary details and focus on the big picture of the system such as system structure, high-level communication protocols, the assignment of software components and connectors to hardware components.***

IndexTerms—ADL,SoftwareTesting, Components, Connector, Composition.

I. INTRODUCTION

The technique is based on software architectures, which specify the primary components, interfaces, connections and configurations of software systems. Software architecture serves as a framework for understanding system components and their interrelationships, especially those attributes that are consistent across time and implementations. This understanding is necessary for the analysis of existing systems and the synthesis of future systems. For this reason, software architecture has drawn wide attention from both academics and industry. At the software architecture level, software systems are presented at a high level of abstraction where a software system is viewed as a set of compositional components, interactions among these components, and the configuration of the system.

Manuscript published on 30 August 2013.
***** Correspondence Author (s)
Bibhuprasad Sahu MTECH(CSE)from NIST,. Currently working as a lecturer in CSE dept of Gandhi Institute For Technology.

II. CURRENT SCENARIO AND NEED OF A FORMAL APPROACH

Although formal unit and module testing criteria have been well studied, system testing is typically done through informally, using manual, ad-hoc techniques. This informality makes it difficult to measure the quality of testing, leads to a lack of repeatability in the process and results, and it means that the tester cannot be confident in the effectiveness of the testing. Unit testing techniques have also sometimes been used to directly generate tests for system-level testing, but there are two problems with this approach. First, this process is simply too expensive to be practical, and second, the kinds of faults that occur at the system level are different from those found during unit testing. Such difficulties can be avoided with the help of software architecture based testing. In this type of testing ,software components can be analyzed and tested independently, low level details can be hidden, which permits concentration to be focused on analysis and decisions that are most crucial to the system structure.

III. METHODOLOGY

The methodology used to perform system testing based on software architecture is the use of formalized software architecture description languages(ADLs).ADLs formally describe how the software system is expected to behave in a high level view that allows test engineers to focus on the overall system structure, and also in a form that can be handled automatically. In ADLs, software architecture is viewed as a combination of components, connectors and configurations. For each component and connector, its interface, types, constraints and semantics are defined. The various terms used in describing software architecture are described below.

3.1 Terms related to components

Components are independent computational elements in architecture.

Component's Interface is a set of interaction points between one component and other components. It specifies the services (messages, operations and variables) a component provides. Interfaces in ADLs are represented as ports.

Component Semantics is a description of component behavior; it enables analysis, constraint enforcement, and mappings of architectures across levels of abstraction.

Component Constraints is a property of or assertion about a system or one of its parts.

3.2 Terms related to connectors

Connectors are architectural units which define the interactions among various component types.

Connector's interface is a set of interaction points between it and the components attached to it. It enables proper connectivity of components and their communication in software architecture.

Connector Semantics provides connector protocol and transaction semantics so as to be able to perform analysis of component interactions, consistent refinements across levels of abstraction, and enforcement of interconnection and communication constraints.

Connector Constraints specifies constraints to ensure adherence to interaction protocols, establish intra-connector dependencies, and enforce usage boundaries.

3.3 Terms related to configurations

Composition ability describes software systems at different levels of detail to support the situations where software architecture may become a mere component in a bigger architecture.

Architecture Configurations Constraints describe desired dependencies among components and connectors in a configuration.

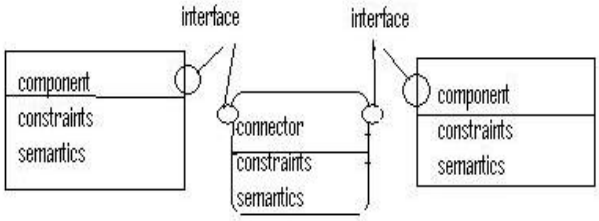

Fig1. Software architectural units and their inter communication

In the above diagram, a component is represented by a rectangular box and a connector is represented by a circular box. Component interfaces and connector interfaces are represented by clear circles on the edges of component and connectors. Each component and connector has its own constraints and semantics. Our software architecture-based testing technique focuses on testing the relations among architecture components. Therefore, we define relations among architecture units as software ***architecture relations***. The technique requires that tests cover certain architecture relations among components and connectors or inside a component and a connector. These architecture relations are based on the possible bindings: data transfer, control transfer, and execution ordering rules. It must be ensured that all the existing architecture relations be covered in the test. The following are some of the testing requirements that must be met at the architecture level:

Testing the connectivity and compatibility from a component interface to a connector interface.Testing the connectivity and compatibility from a connector interface to a component interface.Testing the possible data transfer, control transfer, and ordering relations among the interfaces inside a component.

Testing the possible data transfer, control transfer and ordering relations among the interfaces inside a connector.Testing the connection of two components through a connector.

Other issues that should be included in the testing process at the architecture level are as follows:

Semantics, constraints and interfaces associated with components should be consistent with respect to each other. Component constraints and semantics should have no conflicts. Component constraints and semantics should be deadlock free. Component constraints and semantics should have no conflicts with the component interface constraints. Connector constraints and semantics should have no conflicts.

Connector constraints and semantics should be deadlock free. Connector constraints and semantics should have no conflicts with the associated connector interfaces constraints. Component interfaces should be compatible with the associated connector interfaces. Compatibility means that a component interface behaves in a manner that is consistent with assumptions made by the connector.

IV. CONCLUSION

From the above discussion it can be concluded that testing of software is highly influenced by the software architecture. The use of independent and reusable components in the software architecture will assist in effective system testing performed at the architecture level. Software architecture descriptions have further promising I decreasing the cost and improving the quality of software product. Various architectural views would improve the extend to which one can have a reason about the impact of making change to some artifact of developing process. As

REFERENCES

[1] Mary Shaw "The Coming-of-Age of Software Architecture Research" , Institute for Software Research, International Carnegie Mellon University.

[2] David S. Janzen "Software Architecture Improvement through Test Driven Development" by, University of Kansas.

[3] IanSummerville,"*SoftwareEngineering*",8th Edition, 2007, Pearson Education Inc., New Delhi.

[4] *In International Software Architecture Workshop*, pages 129–132, November 1998.

[5] J.A. Stafford and A.L. Wolf. "Architecture Level Dependence Analysis".

[6] Antonio Bertolino"Software testing research and practice" by, ISTI-CNR, Italy

[7] Roger S. Pressman, "*Software Engineering: A Practitioner's Approach*", 7th International Edition, McGraw-Hill Education (Asia), Singapore

Bibhuprasad Sahu MTECH(CSE)from NIST,.Currently working as a lecturer in CSE dept of Gandhi Institute For Technology.various architectural views would design.

Published By:
Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP)