International Journal of Information & Computation Technology.
ISSN 0974-2239 Volume 4, Number 14 (2014), pp. 1387-1394
© International Research Publications House
http://www. irphouse.com

A Strategic Approach to Software Testing

Keertika Singh[1], Sumit Kumar Mishra[2], Gaurav Shrivastava[3]

[1, 2, 3]*Deptt. Of Computer Science,*
Babu Banarasi Das University (BBDU) Lucknow

ABSTRACT

This paper provide a precise summery of a survey of software testing approach and technique. This reviews some of the common approaches in the integration testing of object – oriented program are also included. It helps in evaluating the effectiveness of testing by providing data on different coverage items. Different keywords were defined as a search string based on the research questions. These include Code Coverage, Software Testing, Prototype Testing, and Traceability. In this paper on the basis of current researches on software testing and preexisting approach of software strategy is described. Software is a critical element of software quality assurance and represents the ultimate review of specification, design and code generation. Software is tested from two different perspectives one, Internal program logic is exercised using 'white box or glass box' test case design technique second, software requirement are exercised using 'black box' test case design technique. This paper provides a study of current test coverage researches conducted by the other researches for test coverage in software testing. Testing is very important activity in software development process. It is to examine and modify source code. This paper deals with a significant and important issue of testing. It is conducted basically manually as well as automated both have their own merits and demerits. Testing automation tools enables developers and testers to easily automate the entire process of testing in software development. This paper includes description about automated Testing tool such as Test Complete.

KEYWORD; SQA, Testing Technique, V&V, Testing Levels, FSM, Test Case Design.

INTRODUCTION:

Today, every company has mechanism to ensure quality by applying solid technical methods are measures, and performing well planed software testing technique. The

study presented here is that *the latest research and issues on software testing* [1].

SOFTWARE TESTING TECHNIQUE:

Fundamental Principal of Testing:

The objective of the testing is to provide a quality product to the customer [2].

1. The goal of testing is to find defects before customers find them out.
2. Exhaustive testing is not possible; program testing can only show the presence of defects, never their absence.
3. Testing applies all through the software life cycle and is not an end of- cycle activity.
4. Understand the reason behind the test.
5. Test the tests first.

Testing Objective:

The main objective of testing is-

1. Testing is a process of executing a program with the intent of finding an error.
2. A good test case is one that has a high probability of finding an as-yet-undiscovered error.
3. A successful test is one that uncovers an as-yet-undiscovered error. [3]

TYPE OF TESTING APPROACH

Verification and Validation approach:

Verification refers to the set of activities that ensure that software correctly implements a specific function. *Validation* refers to a different set of activities that ensure that the software that has been built is traceable to customer requirements [4, 5].

Verification: "Are we building the product right?"

Validation: "Are we building the right product?

Levels of Testing:

Unit level Procedure:

Unit is the smallest part of a software system which is testable it may include code files, classes and methods which can be tested individual for correctness. Unit is a process of validating such small building block of a complex system, much before testing an integrated large module or the system as a whole. Driver and/or stub software must be developed for each unit test

A *driver* is nothing more than a "main program" that accepts test case data, passes such data to the component, and prints relevant results.

Stubs serve to replace modules that are subordinate (called by) the component to be tested. A stub or "dummy subprogram" uses the subordinate module's interface.

Integration Testing:

Integration is defined as a set of integration amoung component. Testing the

interactions between the module and interactions with other system externally is called *Integration Testing.*

Type of Integration Testing

- Top-down Integration
- Bottom-up Integration
- Regression Testing
- Smoke Testing

TOP-DOWN INTEGRATION:

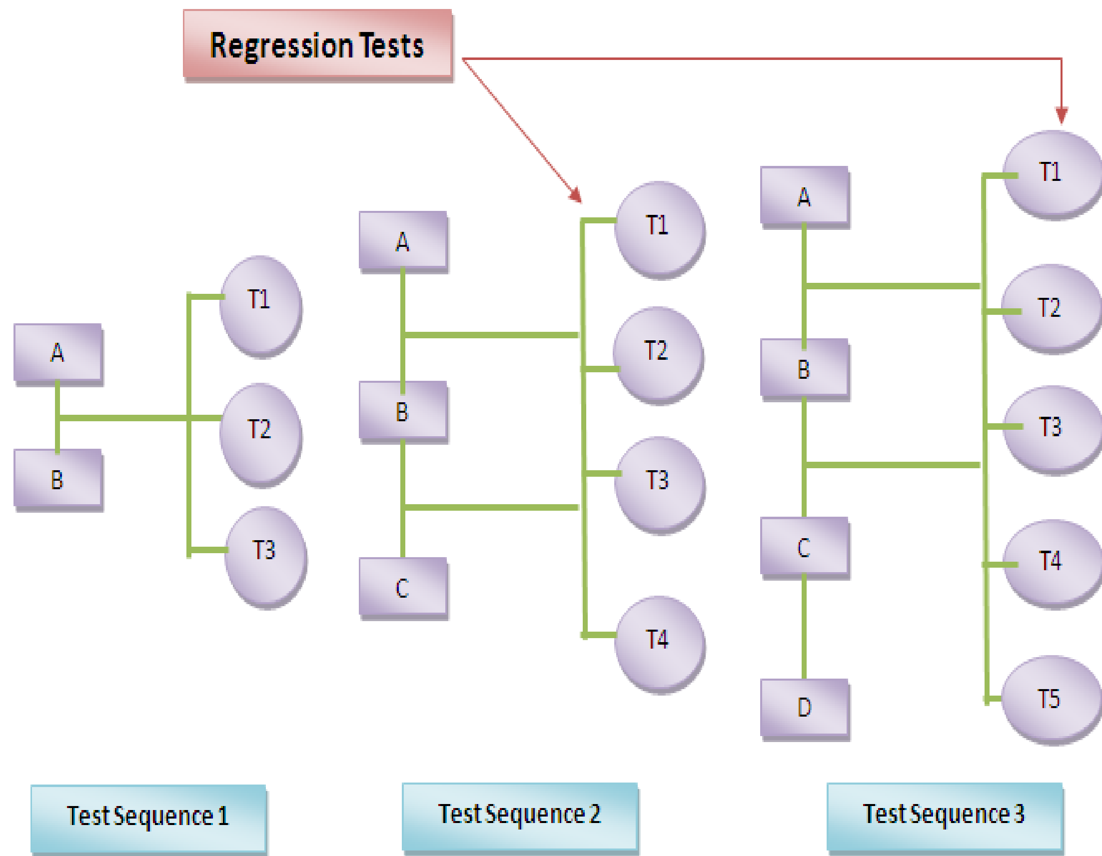
Top-down integration testing is an incremental approach to construction of program structure. Modules are integrated by moving downward through the control hierarchy, beginning with the main control module.

BOTTOM-UP INTEGRATION:

Bottom-up integration testing, begins construction and testing with *atomic modules*. Because components are integrated from the bottom up, processing required for components subordinate to a given level is always available and the need for stubs is eliminated.

REGRESSION TESTING:

Each time a new module is added as part of integration testing, the software changes. These changes may cause problems. In the context of an integration test strategy, *regression testing* is the re-execution of some subset of tests that have already been conducted. Regression testing is the activity that helps to ensure that changes do not introduce unintended behavior or additional errors. Regression testing may be conducted manually, by re-executing a subset of all test cases. [6]



SMOKE TESTING:

Smoke testing is an integration testing approach that is commonly used when "shrink wrapped" software products are being developed, allowing the software team to assess its project on a frequent basis. The smoke testing approach encompasses the following activities: [7]

1. Software components that have been translated into code are integrated into a "build. " A build includes all data files, libraries, reusable modules, and engineered components.
2. A series of tests is designed to expose errors that will keep the build from properly performing its function.
3. The build is integrated with other builds and the entire product is smoke tested daily. The integration approach may be top down or bottom up.

System Testing:

Main Objective	System testing is actually a series of different tests whose primary purpose is to fully exercise the computer-based system.
When to perform system testing	After the integration testing
Who are going to perform	Development team and user
Method	Problem or configuration management
Input	Requirement document, test plan, system test plan
Output	Test report
Tool	Automated tool

INTEGRATION TECHNIQUE

Common Integration Technique:

In this section, we review some of the common approaches in the integration testing of object- oriented programs.

State Based Testing:

State-based testing techniques rely on the construction of a finite-state machine (FSM) or state-transition diagram to represent the change of states of the program under test. For integration testing, the construction of global FSM may become unmanageable and subject to the state-explosion problem. Conventional techniques for concurrent programs treat a program as static set of communicating components and model it as deterministic or non-deterministic FSMs communicating with one another. They systematically arrange the components into an FSM hierarchy by reducing composite FSMs at each level by means of abstraction, and classifying interaction statements as local and global points [8]. Alternatively, they may prune out unnecessary or infeasible state transitions, such as by employing interface processes [9]. Some methods even store the removed details in the edges [10]. The resultant graphs often consist of much fewer nodes and edges than the flattened composition of all FSMs, so that it will be feasible to traverse all nodes and edges. In this way, the state-explosion problem can be alleviated.

Event Based Testing:

Instead of using the state-based approach, the synchronization sequence for a concurrent pro- gram can also be viewed as relationships between pairs of synchronization events. A merit of these approaches is that they support the analysis of event sequencing requirements without having to cater for the states of the program or components.

The instance, relationships between pairs of events can be classified into three types, namely “always valid”, “possibly valid”, and “never valid”. They represent, respectively, situations where the first event should be, may be, and should not be followed by the second event. The “possibly valid” relationship is further classified

into “possibly true” and “possibly false”. Suppose, for instance, it is “always valid” that an event *P* is followed by an event *Q*, and “never valid” that event *Q* is followed by an event *R*. We can conclude that event *P* followed by event *R* is “never valid”. This negative relationship can be used to check the output. A coverage criterion can be specified to cover the “always valid” and “possibly valid” situations for all event pairs identified directly or indirectly, and test output can also be checked for non-violation of the “never valid” situation. Common methods such as random testing and partition testing can be used to generate test cases. Further research will be required to study

Testing Against Formal Specification:

A lot of research has been done using formal specifications for the testing of object-oriented programs at the class level. For example, Doong and Frankl [11] have proposed to test behavioral equivalence of two objects in a class by applying algebraic specification techniques [12]. However, relative little work has been conducted for integration testing.

Contract [13] is a formal language for specifying the behavioral dependencies and interactions among objects of different classes. Such behavioral properties are defined using “message- passing rules”. Testing procedures have been defined for individual mp-rules as well as composite mp-rule. Hence, they have implemented two automatic testing tools for integration testing using Arity/Prolog.

Introducing Automated Testing and Test Complete

Automated Testing is the automatic execution of software testing by a special program with little or no human interaction. Automated execution guarantees that no test action will be skipped ; it relieves tester of same boring step over and over. Developers write unit test cases in the form of little programs, typically in a framework such as JUnit [14].

Test Complete provide special feature for automating test action, creating test, defining baseline data, running test and logging test result.

For example; “Recording Test” feature that create test visually just need to start recording, perform all the needed action against the tested application and test complete will automatically convert all the recorded action to a test

CONCLUSION:

1. The goal of testing is to find defects before customers find them out.
2. Testing applies all through the software life cycle and is not an end of - cycle activity.
3. Understand the reason behind the test.
4. Test the tests first.
5 Tests develop immunity and have to be revised constantly.
6. Defects occur in convoys or clusters, and testing should focus on these convoys.
7. Automated software testing has become necessity of companies because it saves time.

8. test complete is very good tool for automation.

References:

[1] Muhammad Shahid1, Suhaimi Ibrahim and Mohd Naz'ri Mahrin *2011 International Conference on Telecommunication Technology and Applications Proc. of CSIT vol. 5 (2011)*

[2] Software Testing - Principles & Practices *Gopalaswamy Ramesh* and *Srinivasan Desikan*

[3] Myers, G., *The Art of Software Testing,* Wiley, 1979.

[4] *Software Engineering Economics,* Prentice-Hall, 1981

[5] Van Vleck, T., "Three Questions About Each Bug You Find, " *ACM Software Engineering Notes,* vol. 14, no. 5, July 1989

[6] Shooman, M. L., *Software Engineering,* McGraw-Hill, 1983.

[7] McConnell, S., "Best Practices: Daily Build and Smoke Test", *IEEE Software,* vol. 13, no.

[8] S. Lyer and S. Ramesh. Apportioning: a technique for efficient reachability analysis of concurrent object-oriented programs. *IEEE Transactions on Software Engineering*, 27 (1)

[9] S. C. Cheung and J. Kramer. Context con- straints for compositional reachability analysis. *ACM Transactions on Software Engineering and Methodology*, 5 (4)

[10] P. V. Koppol, R. H. Carver, and K. -C. Tai. Incremental integration testing of concurrent programs. *IEEE Transactions on Software Engineering*, 28 (6):607–623,

[11] R. -K. Doong and P. G. Frankl. The ASTOOT approach to testing object-oriented programs. *ACM Transactions on Software Engineering and Methodology*, 3 (2):101–130,

[12] J. A. Goguen and J. Meseguer. Unifying functional, object-oriented, and relational pro- gramming with logical semantics. In *Research Directions in Object-Oriented Programming*, B. Shriver and P. Wegner (eds.), pages 417–477. MIT Press, Cambridge, Massachusetts, 1987.

[13] R. Helm, I. M. Holland, and D. Gangopadhyay. Contracts: specifying behavioral compositions in object-oriented systems. In *Proceedings of the 5th Annual Conference on Object- Oriented Programming Systems, Languages, and Applications (OOPSLA '90), ACM*

[14] JUnit. URL http://junit. org.

ABOUT THE AUTHOR:

SUMIT KUMAR MISHRA received his B. Tech degree from G. B. T. U. in 2013. Currently he is pursuing M. Tech in Software engineering from Babu Banarasi Das University, Lucknow Uttar Pradesh, India.

KEERTIKA SINGH received her B. Tech degree from G. B. T. U. in 2012. Currently she is pursuing M. Tech in Software engineering from Babu Banarasi Das University, Lucknow Uttar Pradesh, India.

GAURAV KUMAR SRIVASTAVA received his B. Tech degree from G. B. T. U. in 2010.

Currently he is pursuing M. Tech in Software engineering from Babu Banarasi Das University, Lucknow Uttar Pradesh, India.