RESEARCH ARTICLE OPEN ACCESS

A Complete Study of Systematic Test Design Techniques throughout Software Development Life Cycle (SDLC)

K. Sai Likhith Reddy*, Dr. K. Sai Prasad Reddy**, Prof. K. Nagabhushan Raju***

**(4th Year B.Tech(CSE), Department of Computer Science, Amrita Vishwa Vidyapeetham, Amritapuri, Kollam-690525, India. Email: ksailikhithreddy08@gmail.com)*
***(Mentor, Centre for Skill Development, Entrepreneurship & Incubation, Sri Krishnadevaraya University – 515003, India, Email: sai.electronics@skuniversity.ac.in)*
**** (Director, Centre for Skill Development, Entrepreneurship & Incubation, Sri Krishnadevaraya University – 515003, India, Email: knrbhushan@skuniversity.ac.in)*

ABSTRACT
Software testing is an assured element of the Software Development Lifecycle. Testing is one of the most important and most widely used approaches for verification and validation. In this paper some of testing procedures and techniques are discussed. This paper focuses on the state of the art in testing techniques. These test design techniques plays vital role in success of testing process. Software Development Team and Software Testing team will implement all applicable test design techniques in order to evaluate the software successfully. By implementing test design techniques Software Development Team & Software Testing Team will design Test Cases. The common two testing methods, White Box Testing & Black Box Testing and their repeatedly used techniques are used in Software Testing Process.

Keywords – Dynamic Testing, Dynamic Test Design Techniques, Static Analysis, Static Testing, Static Test Design Techniques,

Date of Submission: 04-11-2023 Date of acceptance: 17-11-2023

I. INTRODUCTION

Software testing is a procedure which assess the system. Software testing is a union of Static Testing and Dynamic Testing. Static Testing includes inspection and structured peer reviews of requirements, design, and code. Dynamic Testing is performed manually or by automatic means. It helps in verifying that the system satisfies the specified requirements [1]. It also helps in identifying differences between expected and actual results. Measure Quality results in collection of lot of useful data and information about quality of the software, which helps the organizations to make informed decisions about releasing the software. Some of the reasons to test the Software are 1.Software is likely to have faults. 2. To assess the reliability of the software. 3. Failures can be very high expensive. 4. Helps in avoiding sued by customers. 5. To stay in business. This paper focuses on Test design techniques of both Static and Dynamic testing which are popularly and most commonly implemented in testing process.

II. STATIC TEST DESIGN TECHNIQUES

Static Testing consists of the following Test Techniques. These techniques are widely used during Static Testing of the Software[2].

2.1 Reviews

Review is an activity carried out to verify the programming code or documents at different stages in software development process for its completeness, correctness & consistency. This verification is done w.r.t to previous documents in the SDLC or with respect to established standards or norms that have been agreed upon.

General classification of Reviews

Informal Review: Generally it is one to one meeting between the author of a work product and the team peer, initiated as a request of input regarding a particular artifact or problem.

Semiformal Review: It is facilitated by the author of the work product being reviewed. During Semiformal review comments are made at the end or all through. The review comments raised during Semiformal review are captured and published in a review report which is distributed to the participants

Formal Review: Formal Reviews are facilitated by a knowledgeable individual who is called as a moderator. This moderator should not be a project team member or the author of the work product which is under review process. During the formal review the raised issues are captured and published in a formal report and distributed to all participants and Management.

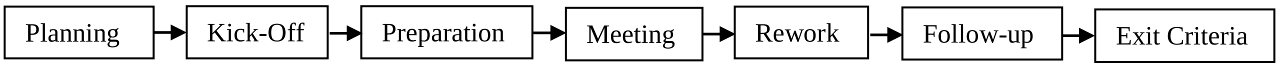


Fig.1 Formal Review Process

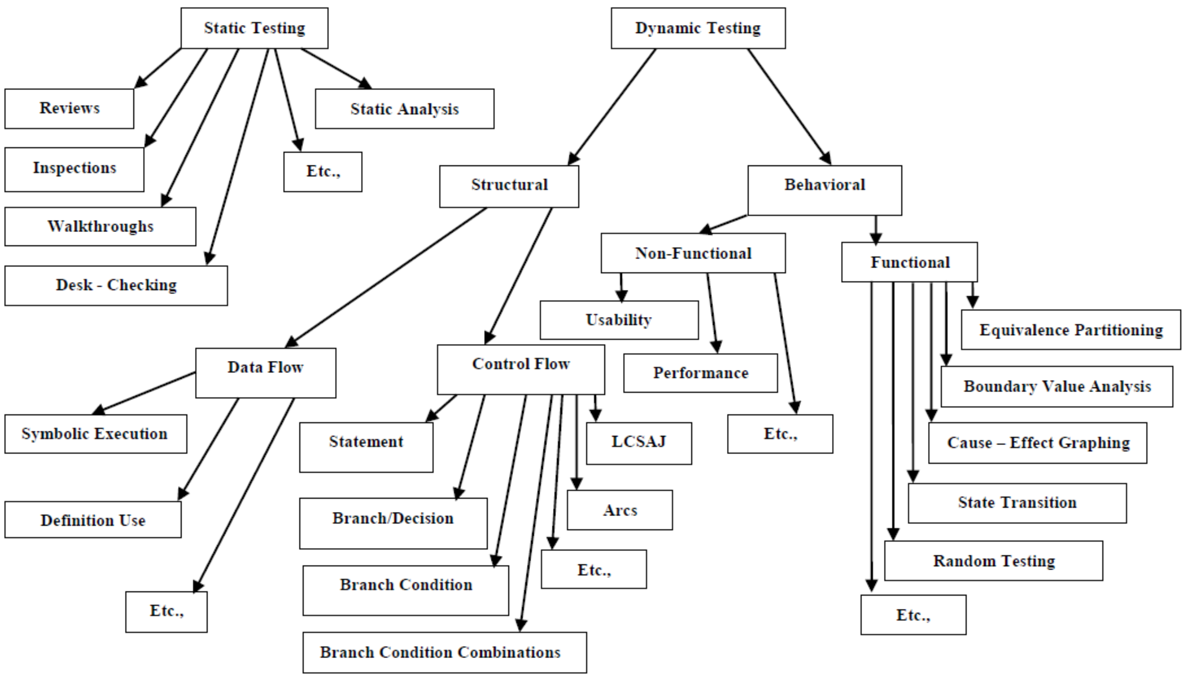


Fig.2: Test Design Techniques

Planning: Awareness of company policies, product requirements and project plans are required for a review to take place. Definition of Entry & Exit criteria and personnel selection is done at this stage.
Kick-off: Moderator make sure that the item is ready for review and distributed. The Moderator also briefs the attendees on their roles and responsibilities.
Preparation: Reviewers will examine the item to be reviewed and also checks for deviation from the standards.
Meeting: Discussions should not be deviated from agenda and checklist. Results will be noted by the scribe. Review comments are raised at the review item.
Re-Work: Scribe prepares document with findings in a Review Summary and submits to Manager. This document contains the defects found and actions of follow-up work to be carried out.
Follow-up: The Moderator ensures that all additional work by the author is checked for completeness and correctness. An additional review may be required; dependant on the amount/complexity of re-works undertaken.
Exit Criteria: It can take the form of ensuring that all actions are completed, or that any uncorrected

items are properly documented, possibly in a defect tracking system.

Advantages of Review

- Reviews are almost 70 percent efficient in finding defect in comparison to testing.
- As defects are identified by the review process in the earlier part of life cycle, they are less expensive to correct.
- Reviews are an efficient method of educating a large number of people on a specific product/project in a relatively short period of time.

Goals of Review

To identify the defects in the early stage of the Software Development Life Cycle i.e in Software requirements stage if self so that defects multiplications can be avoided and defects will not carry forward for the next stages of Soft Development. This is called to as “Phase containment”.

2.2 Walk- through

It’s an informal verification process by which some inputs given, that may or may not be considered for any changes.This is basically carried for obtaining a second person view on the activity carried out. Basically walk–through is done for the source codes

Some sample criteria for Code walk-through

- Minimize or eliminate use of global variables.
- Use descriptive function and method names
- Use both upper and lower case, avoid abbreviations
- Use descriptive variable names
- Use both upper and lower case, avoid abbreviations
- Organize code for readability.
- Use white space generously vertically and horizontally
- Each line of code should contain 80 characters max.
- One code statement per line. Etc,.

2.3 Inspections

Inspection: A formal assessment of a work product conducted by one or more qualified independent reviewers to detect defects, violation of development standards, etc. Inspection will identify defects will not try to correct the defects. Iindividual and group checking uses standards, as per generic and specific rules and checklists by by means of entry and exit criteria. Inspection can able to detect deep-seated faults[3].

2.4 Desk Check

Desk Check is confirmation technique performed by the creator of the project artifact to authenticate the completeness. Desk checking is an casual manual test that programmers can use to verify coding and algorithm logic before a program launch. This allows them to mark errors that may stop a program from working as it should. Desk checking is also known as hand tracing that implies the technique of testing an algorithm's logic and nput/output variables by the programmer. Desk check is executes physically by walking through every line in a pseudo-code to recognize the bugs in logic and to make sure if the algorithm works as proposed. Desk checking helps in locating the bugs or issues in an algorithm before the actual coding and ensures that the code performs as expected. It reduces the time for evaluating the logic while implementing an algorithm to program as the programmer himself checks the logic or syntax errors before moving to later stages.

2.5 Static Analysis

What can static analysis do?

Checks for violations of standards and Checks for things which may be a fault. Descended from compiler technology. A Compiler statically analyses code, and knows a lot about it, e.g. variable usage & finds syntax faults.Static analysis tools can find unreachable code, undeclared variables, parameter type mis-matches, uncalled functions & procedures, array bound violations, etc. Static analysis can find defects without executing the software being examined by the tool. Locates defects that are hard to find in dynamic testing. Developers will use Static Analysis Tools Before and during component and integration testing and When checking-in code to configuration management tools. Designers will use Static Analysis Tools during software modeling.

III. DYNAMIC TEST DESIGN TECHNIQUES

Structural Testing

3.1 Statement Coverage

- Statement coverage is normally measured by a software tool
- In component testing, statement coverage is the assessment of the percentage of executable statements that have been exercised by a test case suite [4].
- The statement testing technique derives test cases to execute specific statements, normally to increase statement coverage

Statement Coverage =

$$\frac{\text{Number of statements exercised}}{\text{Total number of statements}} \times 100\%$$

Example:

Program has 100 statements
Tests exercise 92 statements
Statement coverage = 92%

Example of statement coverage

```
1    read(m)
2    IF m > 6 THEN
3    n = m
4    ENDIF
5    print n
```

Statement Numbers

Test Case	Input	Expected Output
1	7	7

As all 5 statements are 'covered' by this test case, achieved 100% statement coverage

3.2 Decision Coverage

- Decision coverage, related to branch testing, is the assessment of the percentage of decision outcomes (e.g., the True and False options of an IF statement) that have been exercised by a test case suite[6]
- The decision testing technique derives test cases to execute specific decision outcomes
- Branches form decision points in the code

Decision Coverage =

$$\frac{\text{Number of Decision out comes exercised}}{\text{Total number of Decision outcomes}} \times 100\%$$

Example

Program has 80 decision outcomes
Tests exercise 60 decision outcomes
Decision coverage = 75%

3.3 Performance Testing

Performance Testing is done to test the performance of software in the defined conditions. A program/system may have requirements to meet certain levels of performance. For a program, this could be the speed of which it can process a given task. For a networking device, it could mean the throughput of network traffic rate[5]. Often, Performance Testing is designed to be negative, i.e. prove that the system does not meet its required level of performance. The focus of performance testing is on Responsiveness and Scalability.

Behavioral Testing

3.4 Equivalence Class partitioning[7]

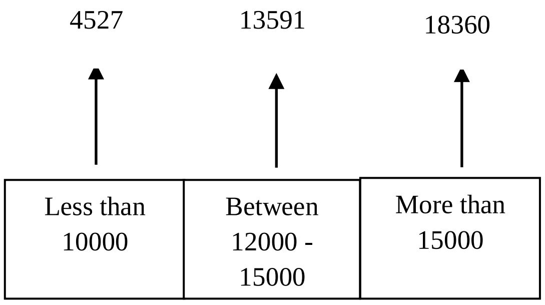
- Equivalence class is a subset of data that is representative of a larger class
- Divide input domain into equivalence classes
- Attempt to cover classes of errors
- One test case per equivalence class, to reduce total number of test cases needed
- Input data and output results often fall into different classes where all members of a class are related
- Each of these classes is an equivalence partition where the program behaves in an equivalent way for each class member
- Test cases should be chosen from each partition

Example

A program which accepts credit limits with a given range Say, $10,000 – $15,000

This would have three equivalence classes

1. Less than $10,000 (Invalid)
2. Between $10,000 and $15,000 (Valid)
3. Greater than $15,000 (Invalid)



Program specification states that the system will accept between 4 and 10 inputs which are 5-digit integers. Partition system inputs and outputs into equivalence sets. If input is a 5-digit integer between 10000 ad 99999, equivalence partitions are <10000, 10000-99999 and >10000

3.5 Boundary Value Analysis

A technique that consists of developing test cases and data that focus on the input and output boundaries of a given function

- Complements equivalence partitioning , selects the test cases at the "edge" of equivalence classes.
- In practice, more errors found at boundaries of equivalence classes than within the classes
- Divide input domain into equivalence classes

Example

The boundary analysis would test
1. low boundary plus or minus one ($ 9,999 and $10,001)
2. On the boundary ($10,000 and $15000)
3. Upper boundary plus or minus one ($14999 and $15001)
Range of Boundary Values
- Value immediately below range
- First value of range
- Second value of range
- Value immediately below last value of range
- Last value of range
- Value immediately above range

Faults tend to lurk near boundaries. Boundaries are good place to find faults. Test values on both sides of boundaries[8]

3.6 State Transition Testing

- State Transition diagrams, like decision tables, are another excellent tool to capture certain types of system requirements and to document internal system design[9]
- State Transition diagrams document the events that come into and are processed by a system as well as the system's responses. They specify very little in terms of processing rules
- When a system must remember something about what has happened before or when valid and invalid orders of operations exist, state transition diagrams are excellent tools to record this information
- These diagrams are also vital tools in the tester's personal toolbox
- State-transition diagrams information can easily be used to create test cases. Four different levels of coverage can be defined[10].
- Create a set of test cases such that all paths are executed at least once under test. While this level is the most preferred because of its level of coverage, it may not be feasible. If the state-transition diagram has loops, then the number of possible paths may be infinite [11].

For example, given a system with two states, A and B, where A transition to B and B transitions to A. A few of the possible paths are:

$A \rightarrow B$
$A \rightarrow B \rightarrow A$
$A \rightarrow B \rightarrow A \rightarrow B \rightarrow A \rightarrow B$
$A \rightarrow B \rightarrow A \rightarrow B \rightarrow A \rightarrow B \rightarrow A$
$A \rightarrow B \rightarrow A \rightarrow B \rightarrow A \rightarrow B \rightarrow B \rightarrow A \rightarrow B$
and so on forever.

Testing of loops such as this can be important if they may result in accumulating computational errors or resource loss.

IV. CONCLUSION

This paper has focuses and presented some of Test design techniques related to Static Testing and Dynamic Testing which will be implemented throughout Software Development Life Cycle. These techniques will be implemented by Software Development Team and Software Testing Team. The importance and advantages of Test design techniques with examples are presented in this paper. For better understanding Test design techniques are presented with examples and pictorial diagrams. Based on the above presentation a number of general conclusions in the context of applying Test design techniques at various stages of Software are presented.

REFERENCES

[1]. J. A. Whittaker, “What is Software Testing? And Why Is It So Hard?” IEEE Software, 2000, pp. 70-79.

[2]. Luo, Lu, and Carnegie, "Software Testing Techniques-Technology Maturation and Research Strategies’, Institute for Software Research International-Carnegie Mellon University, Pittsburgh, Technical Report, 2010.

[3]. Everett et al., “Software testing: testing across the entire software development life cycle”. John Wiley & Sons, 2007.

[4]. J.Irena. “Software Testing Methods and Techniques”, 2008, pp.30-35.

[5]. P. Ron. Software testing. Vol. 2. Indianapolis: Sam’s, 2001

[6]. C. Michael, “Generating software test data by evolution ,Software Engineering”, IEEE Transaction, Volume: 27, 2001.

[7]. R. Ramler, S. Biffl, and P. Grünbacher, "Value-based management of software testing," in Value-Based Software Engineering. Springer Science Business Media, 2006, pp. 225–244.

[8]. Frankl P. and Weyuker E. A formal analysis of the fault-detecting ability of testing methods. Software Engineering, IEEE Transactions. 1993.

[9]. Frankl P. Hamlet D. Littelewood B. and Strigini L., Choosing a testing method to deliver reliability. 19th International conference on Software Engineering, ACM, 1997

[10]. Victor R. Basili and Richard W. Selby. Comparing the effectiveness of software testing techniques. IEEE Transactions on Software Engineering, 13(12):1278–1296, December 1987.

[11]. K. Vijaya Sai Rachana et al., International Journal of Advanced Research in Computer Science and Software Engineering 6(6), June- 2016, pp. 376-380