Software Testing through Evidence Gathering

Priyanka Mathur #1 and Swati V. Chande #2

#1Department of Computer Science, The IIS University, Jaipur, INDIA
#2International School of Informatics and Management, Department of Computer Science, Jaipur, INDIA.

Abstract

An Evidence-based approach is using a best available evidence for making a thoughtful decision about a given set of problem. Evidence-based approach is an amalgamation of individually gained expertise with the evidences gathered through an organized research based approach. Its basic principles are that all practical decisions made should 1) be based on research studies and 2) that these research studies are selected and interpreted according to some specific norms characteristic for Evidence Based Practice[EBP].

Many software testing techniques are proposed to test various types of software and based upon the evidences gathered an effective testing methodology is adopted for a software. This is evidence based approach for software testing.

The software techniques are classified on the basis of three criteria’s a) stepwise code reading b) performing functional testing by adopting boundary value analysis and c) all the statements are covered using structural testing. The study compares the strategies with respect to fault detection effectiveness and fault detection time complexity.

In Evidence-based software engineering (EBSE), all the experiences are properly documented in order to inform software practice adoption decisions. In this research paper, the study factor would be the technology of interest. The technological specifications should be very detailed and not at a very high level of abstraction that is the software lifecycle and all the design methods should be properly read and documented and only then should the engineer collect evidences on it and design the software generation model.

In this paper we analyze the gathered evidence so as to classify the testing strategies on the basis of applicability and types of testing.
Various software testing strategies were studied in which white box testing and structural testing are the most preferred methods when using GA (Genetic Algorithm) and SA (Simulated Annealing) as a technique.

1. Introduction
Meeting the objective of the paper software testing techniques identified are

![Diagram of Testing Techniques](image)

**Fig. 1: Testing Techniques.**

In accordance with the aforesaid objectives, research papers from eminent researchers were studies and Basili (1990) suggested that for the success of a software product software testing and fault detection activities should be exactly and adequately understood, as they are very crucial for the success of a software product. Thus an experimentation methodology is applied to test the software effectively. Some common testing techniques are applied to different types of software and software testing effectiveness is measured on the basis of several factors like:

- Testing technique
- Software type
- Fault type
- Tester experience

And an intercommunication among all these factors.

The most commonly referred software testing techniques are

- Functional testing (black box approach)
- Structural testing (white box approach)
- Code reading

In Code reading identification of subprograms, evaluation of their functionality is done. Further all subprograms are integrated and again their functionality is tested.
Bergstra (2012) suggested a new software testing strategy: Instruction Sequence testing. In the paper the researcher compares testing from the point of view of two different classical definitions of testing. The first definition by King (1976) says that “in testing a small sample of data that the program is expected to handle program is presented to the program. If the program is judged to produce correct results for the sample it is assumed to be correct.” And the second definition as per Singh (2012) is “Testing is the process of executing a program with the intent of finding faults”.

A comparative analysis of the first theory reflects that it’s a program working on a machine that produces output, and there may or may not be any human intervention in testing.

The second definition reflects that a test is successful if and only if it finds faults. The author crafts a term Polinseq which means polyadic Instruction Sequence testing wherein a program is tested instruction wise, marking a difference between program testing and software testing.

The complexity of the program and program testing makes Polinseq as a good testing technique but not a risk free technique.

Harman (2009) ET al. is of the opinion that software testing is the process to measure the quality of developed software. Quality here encompasses completeness, correctness, security and other non functional requirements like reliability, capability, maintainability, efficiency, portability, compatibility and usability.

Evidence based software testing holds a good weightage in this field and more than half the percentage papers are published regarding software testing.

Evidence based or Search based test data generation is the possible number of inputs to the program or test cases and their fitness function. As an example taken in this report to achieve branch coverage the fitness function accesses the closeness of test to executing an uncovered branch; in order to find worst case execution time, fitness is simply the duration of execution for the test case in question.

Gathering data from the research paper presented by Khan(2010) the conclusions drawn are that:

- The main aim of testing can be quality assurance, validation and verification.
- Automated testing can be performed in well controlled softwares.
- A successful testing technique uncovers an undiscovered error.
- Generally software testing is done to affirm the quality of software by systematically testing the software in controlled circumstances.

The testing techniques listed above are suitable for Object Oriented Paradigms. There has been much other work on structural test data generation for the OO paradigm. And to analyze the gathered evidence a comparative study of objective/fitness function and problems faced in are listed in Table 1.
Table 1: Comparative study of objective/fitness function.

<table>
<thead>
<tr>
<th>Testing technique</th>
<th>Technique</th>
<th>Description</th>
<th>Objective /Fitness function</th>
<th>Problem faced</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>Structural Testing</td>
<td>GA, SA (Simulated annealing)</td>
<td>Branch coverage, data flow coverage, decision-coverage.</td>
<td>Maximise path-coverage</td>
<td>tended to avoid the branches that were hard to cover.</td>
<td>Girgis, Xiao [7]</td>
</tr>
<tr>
<td>Structural Testing</td>
<td>GA</td>
<td>Combine non-functional testing goals with coverage based adequacy criterion as a multi-objective problem.</td>
<td>Maximise branch coverage and dynamic memory allocation</td>
<td>Easy to detect faults may become harder to detect when they interact</td>
<td>Lakhotia [8]</td>
</tr>
<tr>
<td>White-box testing</td>
<td>GA</td>
<td>Test data generation</td>
<td>Maximise path coverage</td>
<td>Manual target paths identification requires tester creativity, and more time</td>
<td>Ahmed and Hermadi[9]</td>
</tr>
<tr>
<td>Structural Testing</td>
<td>Genetic algorithms (GAs) and evolutionary strategies (ESs)</td>
<td>Test data generation</td>
<td>Maximise coverage</td>
<td>one parameter may not have effect on another function</td>
<td>Alba and Chicano[10]</td>
</tr>
<tr>
<td>Unit testing</td>
<td>GP</td>
<td>Distance function</td>
<td>Automatic bug fixing</td>
<td>GP is computationally expensive</td>
<td>Arcuri[11]</td>
</tr>
<tr>
<td>Structural Testing</td>
<td>GA</td>
<td>Test data generation for OO software</td>
<td>Maximise data-flow (d-u) coverage</td>
<td></td>
<td>Liaskos[12]</td>
</tr>
<tr>
<td>Model Based Testing</td>
<td>ACO</td>
<td>Automatic test sequence generation</td>
<td>Maximise all-state coverage and feasibility</td>
<td></td>
<td>Li et al.[13]</td>
</tr>
<tr>
<td>Testing Method</td>
<td>Technique</td>
<td>Objective</td>
<td>Strategy</td>
<td>References</td>
<td></td>
</tr>
<tr>
<td>---------------------</td>
<td>----------------------------</td>
<td>------------------------------------------------</td>
<td>-----------------------------------</td>
<td>---------------------------------</td>
<td></td>
</tr>
<tr>
<td>Mutation Testing</td>
<td>Genetic Programming</td>
<td>Generate and evaluate test cases for the mutation testing.</td>
<td>GP is computationally expensive</td>
<td>Emer and Vergilio[14]</td>
<td></td>
</tr>
<tr>
<td>Temporal Testing</td>
<td>Evolutionary Algorithm</td>
<td>Verifying worst/best case execution time</td>
<td>EA alone is not sufficient for a thorough and comprehensive test of real-time systems.</td>
<td>Pohlheim and Wegener[15]</td>
<td></td>
</tr>
<tr>
<td>Regression testing</td>
<td>Manual</td>
<td>Coverage-focused, slicing</td>
<td></td>
<td>Gupta[16]</td>
<td></td>
</tr>
<tr>
<td>Integration testing</td>
<td>Procedural-design firewall</td>
<td>Optimise worst/best case execution time</td>
<td></td>
<td>Leung and White[17]</td>
<td></td>
</tr>
<tr>
<td>Unit testing</td>
<td>Data flow coverage based</td>
<td></td>
<td></td>
<td>Harrold and Soffa[18]</td>
<td></td>
</tr>
<tr>
<td>Regression testing</td>
<td>Modification-focused, minimization, branch and bound algorithm</td>
<td></td>
<td></td>
<td>Fischer, Hartman and Robson[19]</td>
<td></td>
</tr>
</tbody>
</table>

Thus a comparative analysis of the techniques used and the testing strategies used reflect that if more efforts are done on path coverage then White box testing can prove to be very effective.

References


