Chapter 12
Knowledge Engineering Support for Intelligent Software Test Optimization

D. Jeya Mala
Thiagarajar College of Engineering, India

ABSTRACT

Software testing is one of the most important processes in Software Development Life Cycle (SDLC) to ensure quality of software product. A large number of approaches for test data generation and optimization in software testing have been reported with varying degrees of success. In recent years, the application of artificial intelligence (AI) techniques in software testing to achieve quality software is an emerging area of research. It combines the concepts of the two domains AI and Software Engineering and yields innovative mechanisms that have the advantages of both of them. This chapter demonstrates the development of a novel software test sequence optimization framework using intelligent agents based graph searching technique. To ensure the quality, a set of test adequacy criteria such as path coverage, state coverage, and branch coverage have been applied in the approach discussed in this chapter, while achieving optimization.

INTRODUCTION

Software Testing is the process of executing a program or system with the intent of finding errors (Grottke et.al, 2007, Pressman, 2006). Since it typically consumes at least 50% of the total cost involved in software development; an optimization approach is required for testing without compromising quality (Rudalf et.al, 2006). As Phil McMinn briefed in his paper (McMinn, 2004), exhaustive enumeration of a programs input is infeasible for any reasonably-sized program, yet
random methods are unreliable and unlikely to exercise deeper features of software that are not exercised by mere chance.

Software testing is a dynamic activity in which test input generation and execution can be seen as optimization problems. The input domain of most of the programs is likely to be very large. A cost-surface could be formed by applying the cost function to every possible program input. It is this surface which is effectively being searched when attempting to generate test data.

Given the complexities of software systems, it is extremely unlikely that this cost surface would be linear or continuous. The size and complexity of the search space therefore limits the effectiveness of simple gradient-descent or neighborhood searches as they likely to get stuck in locally optimal solutions and hence fail to find the desired test data.

Ultimately, the cost function used for optimization is typically related to the cost and time needed for the testing process, the size of the test suite and the coverage or fault revealing capability of the test cases. The importance of software test optimization has increased in recent years due to customers need for quick delivery of quality software, reduced software development lifecycle, changing markets with global competition and rapid development of new processes and technologies.

Since software testing is a combinatorial problem in which finding solutions at a reasonable computational cost (Harman et al., 2007) is highly complex, it has been classified as NP-complete or NP-hard, or be a problem for which a polynomial time algorithm is known to exist but is not practical. Hence, software test optimization problem is devised as a multi-objective optimization problem and the overall objective function is to minimize the total cost and time needed for the testing process and to maximize the various test adequacy criteria (coverage based, mutation score/fault revealing ability based etc.) to achieve quality. It is formally given in Table 1.

Intelligent software testing techniques support quality assurance by gathering information about the software being studied (McMinn, 2004). Meta-heuristic search techniques are high-level frameworks which utilize heuristics in order to find solutions to combinatorial problems at a reasonable computational cost (Harman, 2007).

The objective of test sequence optimization is to reduce the number of test executions. A test sequence indicates the execution of a sequence of states / blocks of statements in the given software. It usually corresponds to a test path. Assume that, a set of test cases is available in the test case repository. The intention of software testing process is to execute these test cases against the software under test to cover all the states in it. There are two cases to deal with:

1. If some of the states in a test path are infeasible, then even an exhaustive testing cannot cover them; and testing time spent on these states is merely wasted without an indication of infeasible code to the tester.
2. In looping constructs, some of the states are repeatedly exercised in different test sequences with different test cases; and testing time is simply spent without any benefit since they will not reveal any new type of errors.

If the Software under Test (SUT) is small and simple, then a human tester can identify these infeasible test sequences and repeated subsequences by applying intelligence. But, if it is large and

<table>
<thead>
<tr>
<th>Max.</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Coverage (SUT)</td>
<td>(1)</td>
<td></td>
</tr>
<tr>
<td>Fault Revealing (SUT)</td>
<td>(2)</td>
<td></td>
</tr>
<tr>
<td>Min.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cost (Testing Process)</td>
<td>(3)</td>
<td></td>
</tr>
<tr>
<td>Time (Testing Process)</td>
<td>(4)</td>
<td></td>
</tr>
</tbody>
</table>

**Table 1.**
www.igi-global.com/e-resources/library-recommendation/?id=1

Related Content

Modelling Situation Awareness Information and System Requirements for the Mission using Goal-Oriented Task Analysis Approach
www.igi-global.com/chapter/modelling-situation-awareness-information-system/77718?camid=4v1a

Integrating Access Control into UML for Secure Software Modeling and Analysis
www.igi-global.com/article/integrating-access-control-into-uml/39006?camid=4v1a

Demand-driven Development of Service Compositions in Organizational Networks
www.igi-global.com/article/demand-driven-development-service-compositions/48517?camid=4v1a

Performance Evaluation of Secure Key Deployment and Exchange Protocol for MANETs
www.igi-global.com/article/performance-evaluation-secure-key-deployment/52593?camid=4v1a