Chapter 12
Computational Intelligence for Functional Testing

C. Peng Lam
Edith Cowan University, Australia

ABSTRACT

Software testing is primarily a technique for achieving some degree of software quality and to gain consumer confidence. It accounts for 50% -75% of development cost. Test case design supports effective testing but is still a human centered and labour-intensive task. The Unified Modelling language (UML) is the de-facto industrial standard for specifying software system and techniques for automatic test case generation from UML models are very much needed. While extensive research has explored the use of meta-heuristics in structural testing, few have involved its use in functional testing, particularly with respect to UML. This chapter details an approach that incorporates an anti-Ant Colony Optimisation algorithm for the automatic generation of test scenarios directly from UML Activity Diagrams, thus providing a seamless progression from design to generation of test scenarios. Owing to its anti-ant behaviour, the approach generates non-redundant test scenarios.

INTRODUCTION

Software testing is an important technique for achieving some degree of software quality. It accounts for anything between 50 - 75% of the development cost (Hailpern & Santhanarn, 2002). The three main types of activities associated with software testing are: (1) test case generation, (2) test execution involving the use of test cases with the software under test (SUT) and (3) evaluation of test results. A key task associated with test case generation is obtaining an effective test set. The existence and ease of use of a test oracle is a key issue associated with the evaluation of test results. Owing to the immense input space, exhaustive testing is impossible. Thus, test case generation ensuring their adequacy as well their effectiveness in detecting defects in the software is important.

DOI: 10.4018/978-1-60566-758-4.ch012

This is because testing the SUT with an effective test set will imply its correctness over all possible inputs.

Existing approaches for test case design[1] are categorized as black-box, involving the use of some form of specifications, or white-box, where test cases are derived from the logic of the implemented program. Test case generation in black box testing typically involves exercising a set of rules and procedures found in methods such as equivalence class partitioning and cause-effect graphs on the input domain whereas in white box testing it will typically involved finding test data which will execute a specific, yet to be executed, element of the program such as a statement, branch or path. In order to reduce cost, labour and time as well as to improve the quality of the software, any extensive testing would require the automation of the testing process. However, the current status with test automation is that it primarily deals with the automatic execution of test inputs, code instrumentation and coverage measurements. While there are many available commercial test execution tools, few if any of these specifically address the issue of test case design. A formal specification is required for any significant automation in black-box test case generation. The task of test case design is still largely labour-intensive and hence costly and its automation is still very much in its infancy (McMinn, 2004).

Test cases created and selected on the basis of test adequacy criteria are considered to be more effective in discovering faults in a given SUT. Given a testing criterion (e.g. execution of a specific statement in the program), the task in test case generation is to find an input that will satisfy this criterion. However, it may not be possible to determine whether such an input exists. Given limited resources, the application of meta-heuristic techniques to the problem of automatic test case generation is a promising approach that will provide near-optimal solutions. Lam, Robey, & Li (2003) presented a survey for the application of Artificial Intelligence (AI)/meta-heuristics in software testing. McMinn (2004) in a comprehensive survey also presented similar findings, showing that the focus of most existing work in search based software testing involved the use of genetic algorithms (GA) and concentrated on structural testing and little has been done to address functional testing. GAs have also been used in temporal behaviour testing and the SEMINAL Network (Harman & Jones, 2001) has stated that in comparisons with purely random test data generation techniques, approaches incorporating GAs have shown substantial improvements. Other AI techniques used for test data generation included Ant Colony Optimisation (ACO) (Li & Lam, 2005a; Lam, Xiao, & Li, 2007), the AI planner approach (Howe, Mayrhauser, & Mraz, 1997) and Simulated Annealing (SA) (Tracey, Clark, Mander, & McDermid, 2002). Some previous work involving the application of meta-heuristics for functional testing involved the work of Jones, Sthamer, & Eyres (1995) using a Z specification and Tracey (2000) who tested the conformance of a Pascal program to its specification using SA and GA.

Modelling is a common approach for specifying the behaviour of a system. The Unified Modelling language (UML) is a visual modelling language that can be used to specify, construct, visualise and document the software artefacts of a system. It is the de-facto industrial standard, and increasingly software developers are using UML and its associated modelling tools for requirements elicitation, design and implementation of software systems. The advantage of UML is that it is powerful enough to specify a software system's models visually and efficiently. However, as diagrams, its disadvantage lies in its lack of a formal semantics and it is difficult to apply meta-heuristic techniques directly on UML models for test case generation. Given that the UML is increasingly used for modelling software systems, it is important that tools are developed to support automatic test case generation directly from these graphical design artefacts. Existing attempts include UMLAUT