Test Path Prioritization from UML Activity Diagram Using a Hybridized Approach

Gargi Bhattacharjee, VSS University of Technology, Department of Information Technology, Burla, India
Sudipta Dash, Regional College of Management, Department of Computer Applications, Bhubaneswar, India

ABSTRACT
Software testing is regarded as a pivotal approach to realize a high reliable product. To check for the correctness of results, we require appropriate test cases. UML models are largely used to depict the specifications for software development. Test cases are created independently and based on the sequence of occurrence in the diagrams; they lead to corresponding test paths in the program. In this paper, we have analyzed an activity diagram, consisting of concurrent activities, for generating test paths. The obtained test paths are therefore required to be ranked. We have demonstrated that it is conceivable to apply Genetic Algorithm procedures alongside Ant Colony Optimization technique for not only finding the most critical path but also prioritizing the other paths too for enhancing the effectiveness of software testing.

KEYWORDS
Activity Diagram, Ant Colony Optimization, Concurrent Activities, Genetic Algorithm, Path Coverage, Software Testing, Test Case Generation, Test Path Prioritization

INTRODUCTION
Software testing (Kit, 1995; Mathur, 2008) is a critical segment of product development which realizes a high reliability product alongside controlling the quality framework. The principle objective of testing is to develop the trust of the end user in the accuracy of the system. Testing can be viewed as the procedure of distinguishing the contrast between the expected and actual results. It is found that software testing is a time consuming and costly errand. It devours right around half of the software framework resources. Exhaustive testing is unlikely to be carried out since there are no limits on how much we can test. Thus, to limit the process of testing, the concept of testing criteria was laid down. Satisfying the testing criteria puts an end to testing process. With the escalation in complexity and size of software systems, greater time and manpower were consumed for testing. Rich knowledge and experience in the domain of software testing served as a perquisite for software testers carrying out manual testing. Manual testing is so labor-intensive and error-prone that it becomes obligatory to automate the testing techniques. A varied number of testing techniques have been proposed in the literature. They are as follows.

DOI: 10.4018/IJKBO.2018010106
Black Box Testing

Black Box Testing is a type of testing where the software tester is unaware of the internal working of the application. He remains oblivious to the internal architecture of the system and he does not have any access to the source code. Basically, when a software testing professional conducts this test, he interacts with the system’s user interface. He checks for the correctness of the system’s functionalities by providing their corresponding inputs and examining their outputs against a set of expected outputs.

White Box Testing

White Box Testing is a testing technique which requires a detailed knowledge of the internal logic and structure of the code. The tester chooses inputs to exercise paths through the code and determines the appropriate outputs. He verifies the system’s correctness by ensuring that each path or statement of the system gets executed under certain criteria.

Grey Box Testing

Grey Box Testing is an amalgamation of both black box testing and white box testing. As mentioned earlier, in black box testing, the tester remains oblivious of the internal structure of the module being tested while in white box testing; the internal structure is known to him. Here in this technique, the internal structure is partially known. This involves having access to internal data structures, algorithms, databases and the corresponding design documents. These requirements aid them in designing better test cases but the testing is carried out using the interfaces. The tester verifies the correctness of the functionality with a set of appropriate inputs and examining their outputs against a set of expected outputs. Since, this mechanism covers both the features; it has been named as gray box testing. The name originated, as in the eyes of the tester, the software appears as a semi-transparent/gray box where one can partially see. It is best suited for web based applications.

The above-mentioned testing practices serve as the foundation of software testing. The testing process involves three parts: test suite generation, test execution and test evaluation. For conducting effective testing, we need to create a test suite (Jalote, 2008). A test suite is a collection of test cases. A test case is a tuple with three parameters. It comprises of the input given to the system, the expected output from the system and the observed output produced by the given input. A test case also has various additional pieces of information. These optional fields are: a test case ID, test step or the order of execution number, related requirement(s), test category, author, and check boxes. A test suite is said to attain great code coverage if it figures out extreme number of faults with least number of test cases. Software yielding incorrect result may lead to client disappointment and thereby to monetary misfortune. So, to test the software, adequate amount of test cases need to be produced. Test cases are said to play a key role in the process of software testing in determining the quality of software. Therefore, generation and prioritization of test cases turns out to be of a central area of focus for testing researchers. Extensive amount of research has been carried out in the field of test case generation and prioritization and high-quality research yielding good results have been published. Comparing test case generation with the other two parts, it turns out to be more challenging and difficult. With the advancement in technology, testing tools have started to produce test cases with the aim to cover certain coverage criteria, for example, statement coverage, branch coverage, path coverage and so forth. Nonetheless, these tools report issues, with complex software. Earlier the inception of test cases started only after the codes had been written off. Thus, it increased the time devoted for testing. In order to lessen the time span, testers started shifting their attention to software specifications. Software specifications are the main sources for designing test cases. Software specification can be depicted in the form of UML models, formal language specification or natural language description. Researchers and testing professionals found it convenient to design models using UML. Model based testing can be carried out at an initial stage of software development, thereby leading the testers and developers to carry out their work concurrently. The software engineering industry was in desperate
Managing Professions for Knowledge Management
www.igi-global.com/article/managing-professions-knowledge-management/56365?camid=4v1a

Quality Information Concepts, Evolution, and Applications
www.igi-global.com/chapter/quality-information-concepts-evolution-and-applications/205078?camid=4v1a