Chapter 4.14
Object Oriented Software Testing with Genetic Programming and Program Analysis

Arjan Seesing
Enigmatry, The Netherlands

Hans-Gerhard Gross
Delft University of Technology, The Netherlands

ABSTRACT

Testing is a difficult and costly activity in the development of object-oriented programs. The challenge is to come up with a sufficient set of test scenarios, out of the typically huge volume of possible test cases, to demonstrate correct behavior and acceptable quality of the software. This can be reformulated as a search problem to be solved by sophisticated heuristic search techniques such as evolutionary algorithms. The goal is to find an optimal set of test cases to achieve a given test coverage criterion. This chapter introduces and evaluates genetic programming as a heuristic search algorithm which is suitable to evolve object-oriented test programs automatically to achieve high coverage of a class. It outlines why the object paradigm is different to the procedural paradigm with respect to testing, and why a genetic programming approach might be better suited than the genetic algorithms typically used for testing procedural code. The evaluation of our implementation of a genetic programming approach, augmented with program analysis techniques for better performance, indicates that object-oriented software testing with genetic programming is feasible in principle. However, having many adjustable parameters, evolutionary search heuristics have to be fined-tuned to the optimization problem at hand for optimal performance, and, therefore, represent a difficult optimization problem in their own right.

DOI: 10.4018/978-1-61350-456-7.ch4.14

Copyright © 2012, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.

INTRODUCTION

Testing is the most widely used and accepted technique for verification and validation of software systems. It is applied to measure to which extent a software system is conforming to its original requirements specification and to demonstrate its correct operation (IEEE, 1999). Testing is a search problem that involves the identification of a limited number of good tests out of a shear, nearly unlimited number of possible test scenarios. "Good tests" are those runtime scenarios that are likely to uncover failures, or demonstrate correctness of the system under test (SUT). Identifying good test cases typically follows predefined testing criteria, such as code coverage criteria (Beizer, 1990). This is based on the assumption that only the execution of a distinct feature, or its coverage, can reveal failures that are associated with this feature.

Because the primary activities of testing, test case identification and design, are typical search problems, they can be tackled by typical search heuristics. One of the most important search heuristics for software testing is known to be random testing. This is also one of the most commonly used testing strategies in industry today. Currently, also more advanced heuristic search techniques are applied to software testing. These are based on evolutionary algorithms, and they have also made their ways into industry (Baresel, 2003; Buehler & Wegener, 2003) since their performance in devising test cases was found to be at least as good as random testing, but usually much better (Gross, 2003; Tracey, Clarke & Mander, 1998). The group of these testing techniques is referred to as evolutionary testing (ET) according to Wegener and Grochtmann (1998).

ET is an automatic test case generation technique based on the application of evolution strategies (Schwefel & Männer, 1990), genetic algorithms (Goldberg, 1989; Holland, 1975), genetic programming (Koza, 1992), or simulated annealing (von Laarhoven & Aarts, 1987). ET searches for optimal test parameter combinations that satisfy a predefined test criterion. This test criterion is represented through a "cost function" that measures how well each of the automatically generated optimization parameters satisfies the given test criterion. For a test, various test criteria a perceivable, according to the goal of the test, such as how well a test case covers a piece of code, in the case of structural testing (Jones et al., 1996; Pargas et al, 1999), or how well a test case violates a (safety) requirement (Tracey et al, 1999), for example.

Evolutionary testing has initially only been applied to traditional procedural software. Here, ET is used to generate input parameter combinations for test cases automatically that achieve, i.e., high coverage, if the test target relates to some code coverage criterion. However, also object-oriented software testing with genetic algorithms has been tackled by researchers, e.g., Gross and Mayer (2002, 2003), Tonella (2004), and applying genetic programming approaches, e.g., Seesing and Gross (2006), Ribeiro, Zenha-Rela and de Vega (2008), and Gupta and Rohil (2008).

The main differences of object technology compared to the procedural paradigm are (1) that it is inherently based on states which are not readily visible outside of an object's encapsulating hull, and (2) that an object test, as the basic unit of testing, can incorporate an arbitrary number of operation invocations, or any arbitrary sequence or combination of method invocations.

An object's internal state depends on any previously performed operation invocations, the so called invocation history (Gross, 2005; Gross & Mayer, 2003), including input parameter settings. Hence, object testing involves not only the generation of suitable input parameter combinations for a single procedure under test, but, additionally, the generation of suitable test invocation sequences of various operations of an object, plus the generation of their respective input parameter combinations. As a consequence, in object testing, we have to deal with a number of test artifacts, such as the

13 more pages are available in the full version of this document, which may be purchased using the "Add to Cart" button on the publisher's webpage:
www.igi-global.com/chapter/object-oriented-software-testing-genetic/62493

Related Content

SoC Self Test Based on a Test-Processor
Tobial Koal, Rene Kotheand Heinrich Theodor Vierhaus (2011). *Design and Test Technology for Dependable Systems-on-Chip (pp. 360-376).*
www.irma-international.org/chapter/soc-self-test-based-test/51409

Detection and Classification of Leaf Disease Using Deep Neural Network
Meeradevi, Monica R. Mundadaand Shilpa M. (2022). *Deep Learning Applications for Cyber-Physical Systems (pp. 51-77).*
www.irma-international.org/chapter/detection-and-classification-of-leaf-disease-using-deep-neural-network/293122

Fuzzy Translation of Doubt Interval-Valued Fuzzy Ideals in BF-Algebras
Tripti Bejand Young Bae Jun (2020). *Handbook of Research on Emerging Applications of Fuzzy Algebraic Structures (pp. 225-243).*
www.irma-international.org/chapter/fuzzy-translation-of-doubt-interval-valued-fuzzy-ideals-in-bf-algebras/247657

SCIPS: Using Experiential Learning to Raise Cyber Situational Awareness in Industrial Control System
Allan Cook, Richard Smith, Leandros Maglarasand Helge Janicke (2018). *Cyber Security and Threats: Concepts, Methodologies, Tools, and Applications (pp. 1168-1183).*
www.irma-international.org/chapter/scips/203553

Viewpoint-Based Modeling: A Stakeholder-Centered Approach for Model-Driven Engineering
Klaus Fischer, Julian Krumeich, Dima Panfilenko, Marc Bornand Philippe Desfray (2018). *Computer Systems and Software Engineering: Concepts, Methodologies, Tools, and Applications (pp. 679-704).*
www.irma-international.org/chapter/viewpoint-based-modeling/192898