Class Level Test Case Generation in Object Oriented Software Testing

N. Gupta, Birla Institute of Technology, India
D. Saini, King Saud University, College of Science, Saudi Arabia
H. Saini, Higher Institute of Electronics, Libya

ABSTRACT

Object-oriented programming consists of several different levels of abstraction, namely, the algorithmic level, class level, cluster level, and system level. In this article, we discuss a testing technique to generate test cases at class level for object-oriented programs. The formal object oriented class specification is used to develop a test model. This test model is based on finite state machine specification. The class specification and the test model is analyzed to select a set of test data for each method of the class, and finally the test cases can be generated using other testing techniques like finite-state testing or data-flow testing.

Keywords: class level testing; state space partition; test case generation; test model

INTRODUCTION

The object-oriented technology has many benefits for parts of the entire software development cycle (analysis, design and implementation phases). The object-oriented development process is iterative, the object-oriented paradigm emphasizes reuse, and the items of interest are always the objects. Thus, engineers and managers want to use this technology in their own field. But, for critical systems, which need a certification, the testing process is an important task. Testing can be done at four different levels of abstraction found in object-oriented systems. These are the method level, class level, cluster level, and system level. The method level considers the code for each operation in a class. The class level is composed of the interactions of methods and data that are encapsulated within a given class. The cluster level consists of the interactions among cooperating classes, which are grouped to accomplish some tasks. The system level is composed of all the clusters (Smith & Robson, 1992).

Testing at the method and system levels is similar to conventional program testing. Many researchers have addressed the class-level testing (Doong & Frankl, 1991, 1994; Harrold, McGregor, & Fitzpatrick, 1992; McGregor & Korson, 1994; Murphy, Townsend, & Wong, 1994). Formal methods can be used in software
testing process to improve the quality and effectiveness of the process (Bernot, Gaudel, & Marre, 1991; Carrington & Stocks, 1994; Crnkovic, Filipe, Larsson, & Lau, 2000; Eden & Hirshfeld, 2001). Based on the formal methods we have developed a testing process for object oriented software. In this article we present an object oriented software testing process which can be used to generate test cases systematically and effectively. To generate the test cases for a class, which may have mutable objects (which can be modified after it is created), we will use specification-based testing technique. The purpose of specification-based testing is to derive testing information from a specification of the software under test, rather than from the implementation.

There are various methods of object-oriented software specifications. These may be graphical techniques, decomposition specification techniques, communication specification techniques, functional specification techniques and behavior specification techniques (Wieringa, 1998). In this article we will focus on finite state diagram which is a behavior specification technique, for the generation of test cases. We will consider that classes are instantiated into mutable objects.

CLASS LEVEL TEST CASE GENERATION
A software testing model summarizes how you should think about test development. It tells you how to plan the testing effort, what purpose tests serve, when they’re created, and what sources of information you use to create them [17]. Here, we have extracted our test model from the formal specification. In this strategy, class specification is used to obtain class state space, which is partitioned into substates. A test model is composed of a set of states and a set of transitions among the states. Each state is obtained through the state space partition of the class. Each transition consists of a method, which can change the value of an object from source state to target state. The input space of each method, which is the sets of values for the input parameters of the method, is partitioned. The input space partition values are used with test model to obtain the test data. Finally this test data can be used for the generation of test cases. The process of generating test cases at the class level is illustrated schematically in Figure 1.

CLASS SPECIFICATION
Larch (Guttag, Horning, Garland et al., 1993; Wing, 1983) may be thought of as an approach to formal specification of program modules. This approach is an extension of Hoare’s ideas for program specification (Hoare, 1969, 1972). Its distinguishing feature is that it uses two “tiers” (or layers). A class specification will consist of two layers: a functional tier and a conditional tier. The functional tier is an algebraic specification, which is used to define the abstract values of objects. An algebraic specification may

Figure 1. Process for generating test data
7 more pages are available in the full version of this document, which may be purchased using the "Add to Cart" button on the product’s webpage:
www.igi-global.com/article/class-level-test-case-generation/2644?camid=4v1

www.igi-global.com/e-resources/library-recommendation/?id=2

Related Content

A New Method for Building Probabilistic Ontology (Prob-Ont)
www.igi-global.com/article/a-new-method-for-building-probabilistic-ontology-prob-ont/176906?camid=4v1a

A Semantic Web-based Approach for Context-Aware User Query formulation and Information Retrieval
www.igi-global.com/article/semantic-web-based-approach-context/2638?camid=4v1a
‘Talking Tools’: Sloyd Processes Become Multimodal Stories with Smartphone Documentation
www.igi-global.com/chapter/talking-tools/137421?camid=4v1a

A Multi-Agent Temporal Constraint Satisfaction System Based on Allen's Interval Algebra and Probabilities
www.igi-global.com/chapter/multi-agent-temporal-constraint-satisfaction/5027?camid=4v1a