Abstract

Both the formal and the informal methods should be effectively used to validate object oriented software. We first consider applying conventional approaches to validate OOS and then investigate new aspects caused by the properties of OOS.

1 Introduction

Object oriented programming has been around for about a decade but we have relatively limited experience of object oriented software development from the viewpoint of software engineering.

One of the important issues is verification and validation of object oriented software. Various V & V methods and tools have been invented and used for the conventional programming practices. Are they applicable to OOS? Should we explore new approaches based on the characteristics of OOS?

As every new technology is continuous from the old and revolutionary at the same time, the answers to the above two questions are naturally yes to both. By the same token, if we divide the existing V & V technologies to two categories, formal and informal, we should need both and have to find a good way of integrating them.

2 Conventional Methods

I make brief comments to some of the conventional methods useful and actually being used to OOS.

1. Review

Reviewing object oriented problem domain models, designs, and programs are practical and useful activities. Although the term “formal” review (or walk-through or inspection) is sometimes used, this method is essentially informal. The way of reviewing object oriented products is basically the same as reviewing software of other types, except that you need the insight and knowledge of object oriented technology.

2. Testing

Testing is still the most practiced and effective method of enhancing program quality. The high modularity of OOS may make the module testing and the integration testing easier. On the other hand, inheritance or delegation may make the module/integration testing complicated.

3. Static analysis

Simple static analysis such as interface check of message passing, name consistency checking, etc. should be useful. This kind of technique is one of the most primitive but often useful formal methods.

4. Reuse

Object oriented technology is expected to promote software reuse. Reusing quality assured objects stored in libraries will certainly enhance total quality of software. The fundamental problem of how to supply high-quality reusable objects still remains. There are some experimental works on reusability of OOS (e.g. [1]).

3 New Aspects

As often pointed out, there are several streams that have been flowing into the current object oriented technology: simulation modeling (e.g. Simula), concurrent process modeling (e.g. Actor!), abstract data types, semantic data modeling (e.g. ER model), and knowledge representation (e.g. frame).

The properties of OOT inherited from these preceding streams can be categorized into two major classes: static and dynamic. As new aspects of V & V caused by object oriented software, I focus on the following two issues, each representing typical property of the static aspect and the dynamic aspect. The approaches are both formal.

1. Abstraction hierarchy consistency

By introducing object classification mechanism, abstraction level hierarchy can be naturally modeled by OOT. Assuring the consistency of this abstraction hierarchy is an important problem. For this purpose, formal methods devised for theoretically treating abstract data types, such as algebraic specifications and institution, are now accessible and should be the most powerful tools. There remain works of applying these theories to fit into the object oriented framework.
2. Concurrent behavior correctness

There are a number of concurrent object oriented models and programming languages proposed. It is natural to interpret objects as agents acting concurrently and coordinating by message exchanging mechanism. It is a challenging problem to construct a method of assuring the correctness of the whole system dynamic behaviors. One of the most promising methods is to interpret the semantics of a concurrent programming language and thus the behaviors of programs written in that language by transforming language constructs into well defined formal concurrent system model such as CCS, CSP, Actor, and GHC. There are some pioneer works but good research opportunities exist.

4 Conclusions

Object oriented software validation can be based on conventional approaches but that is not enough. There remains a large field to be studied, including theoretical treatment of object abstraction hierarchy and concurrent system behaviors.

References