Developing Test Adequacy Criteria for OO Software

Brigid Haworth
Bournemouth University
brigid.haworth@fen.baynet.de

Introduction

Structural coverage criteria can be used in the management and control of software testing. Criteria may be used for the design of test cases and test data selection or for code coverage measurement and analysis of test suites. In this work, the development of criteria is based on structural coverage requirements for object oriented software. The focus of this report is on the development of criteria for objects. Related work on the development of criteria for specification based coverage using state machines and ADT graphs has been published in [3]. Criteria based on inter-method message sequences are also explored in [4].

Developing Adequacy Criteria

Adequacy questions arising for object level testing are

- how to adequately test an object
- how to avoid over-testing inherited features in an object

In order to develop criteria, the structure of an object is considered. An object comprises methods and data in a unit of test. Object members may be inherited or locally defined. Connections represent interactions between methods and data and also between methods and other methods. Object errors may arise due to

- incorrect method implementations
- invalid interactions between methods both via direct calls and indirectly via unexpected data values

A 3-level model for testing OO software has been defined [1]. These levels correspond to object level testing, class level testing and cluster level testing. The focus of the object level testing is on the dynamic view. This includes control flows within methods and both control flows and data flows between methods. These flows capture the interactions occurring within and between methods. A unified flow based model for objects is determined by the object level test model. Object flows are defined by triples <M1,C,M2> where M1 and M2 are method points and C represents a flow connection between these points. A grammar defines the types of flows that can occur within each level of the test model. In addition legal categories are defined to restrict the types of flows that are represented in the grammar to those that can occur in programming language.

The object level flow types are;

```
<a,d,a> i.e.
  ancestor_m_point 'direct' ancestor_m_point
<a,d,l> i.e.
  ancestor_m_point 'direct' local_m_point
<l,d,a> i.e.
  local_m_point 'direct' ancestor_m_point
<l,d,l> i.e.
  local_m_point 'direct' local_m_point
<a,a,a> i.e.
  ancestor_m_point ancestor_d ancestor_m_point
<a,a,l> i.e.
  ancestor_m_point ancestor_d local_m_point
<l,a,a> i.e.
  local_m_point ancestor_d ancestor_m_point
<l,a,l> i.e.
  local_m_point ancestor_d local_m_point
<l,l,l> i.e.
  local_m_point local_d local_m_point
```

In abstract terms the test model for an object is viewed as a set of flows. The flows may be categorised in a way that aids the tester in the determination of test requirements for an object with inherited tested features. This shows the tester which of these inherited features may require further testing due to interaction with newly defined features. For example, the flow type <l,a,a> indicates that ancestor data is defined by a method in the derived class and subsequently used by a method in the parent class. This kind of interaction between the methods may be error prone if the method in the derived class does not observe some behavioural constraints expected by the parent method i.e. the derived method may set values in the data variable that are not handled by the parent method. In this case it is necessary to test the interaction of the new method with the parent method. The extent to which inherited object features are "preserved" i.e. are embedded as is, can also be shown by these categories. This information may be regarded as an indicator of the "testability" of the implementation.

A hierarchy of criteria have been defined for object structural coverage. These are

- intra-object method-method call flow coverage
- intra-object branch(decision/condition) flow coverage
- intra-object method-method indirect flow coverage
- intra-object all flows coverage

These criteria are formally defined in [2]. The hierarchy of criteria can be further refined by the use of the types of flows for each criterion as shown in Figure 1. It should be noted that the subsumes relation between the indirect flow coverage and the branch flow coverage holds only in the case where data references are occurring on the branch. This limitation can be overcome for measurement purposes by appropriate instrumentation of the source.

Case Studies

During the development of the test model and associated criteria, two studies were undertaken. These

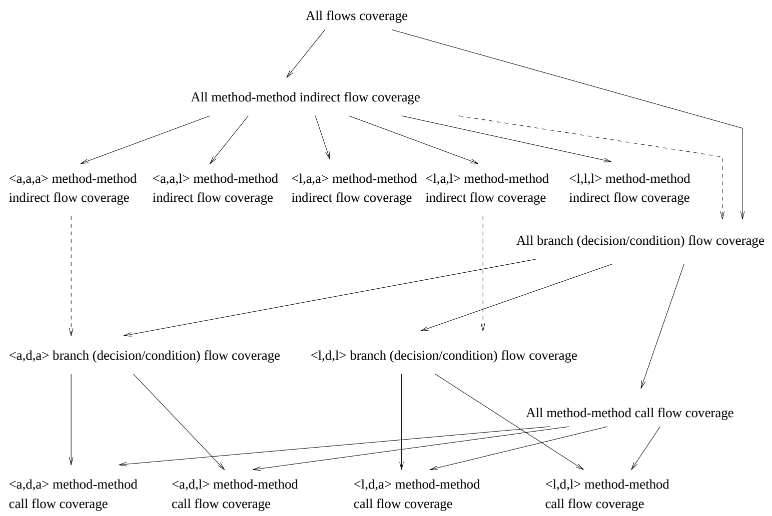

Figure 1: Object Level coverage criteria - subsumes hierarchy

studies were used to support the theoretical development of the criteria. Due to the lack of tool support extensive study is currently infeasible. The first case study was based on an application developed in Delphi programming language. The results are published in detail in [1]. 7 classes forming a cluster were analysed. The static analysis showed that the structure of classes in terms of indirect interactions appears to be independent of the number of methods in the class. The following table summarises the number of methods per class and the types of indirect flows found.

class	method count	<l,l,l>
A	1	0
B	17	99
C	0	0
D	4	1
E	7	8
F	2	1
G	6	23

Also, in this study, 50% of methods in each class had a cyclomatic count of 1 or 2. It is noted however, that the sample is too small to conclude that this is typical of OO software. The results show that there are interactions occurring between methods that cannot be detected using current control flow based techniques. Following this study, the test model was refined in order to improve the level of granularity at which the flows are detected. In the first study, flows between methods were detected on the level of methods. The current model can be used to detect flows between specific points within methods. This refinement was needed in order to define the flow based criteria in such a way that the coverage within methods could be detected. The second study was based on an application developed in C++. A cluster of 5 classes was analysed. The number of methods with types of indirect flows per class are summarised below;

class	method count	<l,l,l>	<a,a,l>	<l,a,l>
A	4	7	0	0
B	20	144	0	0
C	10	206	29	12
D	15	53	0	0
E	3	22	0	0

The object represented by the class C (derived from B, B derived from A) has 179 flows due to branching, has 16 flows due to direct method-method interactions and has 365 flows due to indirect method-method interactions. (33 flows are blocked due to scoping) The object has 34 functions, 2 of which are blocked due to scoping in the hierarchy. The cyclomatic counts for the methods in the classes varied from 1 to 13. Once again, the sample is too small to build a characterisation.

Conclusions

From the studies undertaken, it is clear that control flow techniques are insufficient for detecting method interactions in OO software. Tool support for detecting method interactions due to indirect flows is needed. Profiles based on the interaction types can be used to guide the testing process - the typing information can be used to determine when re-testing of inherited features is necessary. Future work includes the investigation of the relationship between the criteria proposed and state based techniques for test case design. Further empirical evidence to support the approach can be gathered when tool support for measurement is available.

References

[1] B. Haworth, C. Kirsopp, M. Roper, M. Shepperd and S. Webster. Towards the development of adequacy criteria for object-oriented systems. In *Proceedings of the 5th European Conference on Software Testing Analysis and Review*, pages 417–427, Edinburgh, Scotland, Nov. 1997.

[2] B. Haworth. Adequacy Criteria for Object Testing *In Proceedings of the 2nd International Software Quality Week Europe 1998*, Brussels, Belgium, Nov 1998.

[3] A. Spillner. Four Kinds of Class Modality Four Kinds of Class Testing? *In Proceedings of the 6th European Conference on Software Testing Analysis and Review*, Munich, Germany. Nov/Dec 1998.

[4] M. Winter. Managing Object-Oriented Integration and Regression Testing *In Proceedings of the 6th European Conference on Software Testing Analysis and Review*, Munich, Germany. Nov/Dec 1998.