Error-Based Validation Completeness

W.E. Howden

Department of Computer Science
University of California at San Diego
La Jolla, CA 92039

Testing and analysis methods have either an explicit or implicit notion of completeness built into them. Sometimes this is directly related to the correctness or reliability of the program, and sometimes it is not. Coverage measures, for example, are based on a notion of completeness which is only indirectly related to correctness.

We distinguish between two kinds of validation completeness: developmental and acceptance. The only thing that is known for certain about the correctness of a system is its level of confidence, during acceptance testing. Most testing and analysis methods seem to be oriented towards developmental testing. Fault-based testing, for example, looks for the occurrence of errors that manifest themselves in instances of well defined program fault classes which can be effectively detected using tests from test classes associated with the fault classes. It is the basis of, for example, mutation testing, weak mutation testing, and the recent work of Morell and Richardson[1-3].

In the error-oriented approach the goal is to discuss errors directly. Models of errors are devised and validation methods developed which are directly related to those models. Completeness of developmental testing occurs when software has been analyzed for the effects of all errors in the error model. Different validation methods, such as fault-based testing, or proofs of correctness, are incorporated as techniques appropriate for detecting errors in different kinds of reasoning processes.

A simple model which has been effectively used to develop several software analysis tools views programming as the process of constructing a formal representation of an object. Abstraction and decomposition are used to handle complexity. Errors are associated with these mental processes and classified as abstraction or decomposition errors.

In a decomposition error, the programmer, working in one part of a system, makes false assumptions about properties of other parts. Errors can be dealt with by requiring that programmers document these assumptions, and document when they know these properties to be established. Analysis of these comments can then be used to reveal decomposition errors. In the flavor analysis approach, information about the kinds of data manipulated in the program is documented using flavor comments. These associate data flavors with abstract data objects. Other kinds of analysis use comments which document when abstract operators are being performed or are being assumed to have been performed[4].

In an abstraction error there is a mismatch not between mental abstractions, as in decomposition errors, but between some piece of code and the abstraction used by the programmer in his thinking about the code. The effects of errors of this type can be detected using a variety of methods. For example, symbolic evaluation can be used to see if a sequence of simple computations implements a complex symbolic expression.

In the methods we have developed for detecting decomposition and abstraction errors, the user must document the abstractions which are the basis of his or her reasoning processes during programming. This kind of information is often part of the documentation for a program. By giving it a more specific and well defined role, we are encouraging its retention. It is asserted that this information is as important as the code itself, that it is part of the program, and to require its inclusion is not an extra burden on the programmer. It is also vital for effective maintenance and testing. Complete validation is not possible without it, since if it is not available, it is impossible to carry out testing and analysis for errors which involve it, except as a coincidental side effect of validation methods which focus on other kinds of information.

Our position is that in developmental testing the goal should be completeness with respect to programming errors. This results in a paradigm for the integration and use of existing methods, and a basis for the development of new technology, such as comments analysis for decomposition errors. The idea of combining different methods, has been proposed before[5]. The error-based approach facilitates that goal by providing a strategy for integration.