1 Motivation

Software reliability is a measure of how well a program will perform in its operational environment. Researchers in Software Engineering have spent efforts to develop methods and tools for use in different phases of the life cycle. For example, design tools aid in the systematic design of the software in the hope that doing so will improve software quality. We believe that the use of disciplined practices in each phase of the life cycle is necessary, but not sufficient, to achieve high quality. Eventually one needs to resort to testing methods to gain confidence in the software. Our research has concentrated on the development and perfection of methods and tools for use during the testing and maintenance phases.

A variety of software testing techniques have been proposed by researchers during the past two decades. However, the state of practice continues to lag behind the state of the art. Even though researchers have demonstrated the effectiveness of these methods, at least for classes of relatively small programs, and also constructed prototype tools, one reason or the other is considered responsible for this lag. We have identified two problems that need to be overcome in order that some of the powerful testing techniques be used in practice. These are (1) performance and (2) effectiveness. The testing methods that we refer to are data flow and mutation testing.

2 Performance Enhancement

Mutation testing has been empirically found to be the most effective of the software testing methods proposed in detecting faults. It is known to be effective in detecting both faults of omission, e.g. missing assignments or conditions and faults of commission, e.g. incorrect arithmetic formula[5]. However, mutation suffers from the problem of performance. Even though the complexity of mutation is $O(n^2)$, where $n$ is the number of variable references in the program, the number of mutants generated can be prohibitively large even for programs that are less than 100 K lines of code.

It has been our strategy to attack the performance problem using parallel machines. We have attempted to exploit three different architectures for this purpose: (1) vector machines, such as the Cray-Y/MP[9], (2) SIMD machines[7] such as the Connection machine, and (3) MIMD machines such as the Ncube/2. We built a tool named PMotha[3] as a back-end to an existing tool named Motha[2], to perform mutation testing on a 128 processor hypercube. Our experience with this tool was mixed. We found that for some programs which, on the average, compile relatively faster than they execute, our approach used in PMothra worked well. However, for programs that compile slowly as compared to their execution time the utilization and speedup on a parallel machine can be poor. This observation led to the development of an entirely new approach to mutation testing in particular, and testing in general, namely Compiler Integrated testing (CIT). The CIT approach envisages the integration of testing methods into a compiler. This enables the compiler to generate mutants efficiently and improve the utilization of of the parallel machine[4]. We hope that CIT and parallel machines together can make mutation testing a practical reality.

3 Effectiveness

The fault detection effectiveness of mutation testing has been compared in the past by Budd[1]. This study clearly showed mutation to be superior to other white-box methods. Recently we examined and categorized Knuth's errors[6] log of the errors he found while developing TeX. We then sampled some of the “complex” errors in Knuth's log and found that indeed mutation would guarantee the detection of these errors. In yet another study we have found mutation to be signif-
icantly superior to data flow in both its fault detection effectiveness and the strength of the test adequacy criteria[8].

Thus we have shown that mutation is an effective method for testing software to obtain high reliability. We are currently experimenting with a new variant of mutation testing namely constrained mutation which has complexity linear in program size but might have better fault detection capability than data flow. We expect constrained mutation to be a cost-effective alternative to mutation and data flow in applications that do not involve life-threatening software.

4 Reliability Estimation: New Approaches

Until now software reliability estimation has been carried out using approaches similar to the ones used for hardware reliability estimation. These methods make assumptions that are often unrealistic, e.g. new faults are not introduced during debugging. Further, they are time dependent and can therefore provide estimates that depend on the ordering of input data. Using mutation and data flow testing, we have proposed a new model named White-Box model, to estimation software reliability that is free from such disadvantages. Our model is based on program structure and the notions of fault reachability, origination, and propagation. We are currently experimenting with this model to determine how accurate are the estimates obtained by this model.

References


