```
****************************************
*                                      *
*  REPORT ON FLORIDA TESTING WORKSHOP  *
*            Richard Hamlet            *
*                                      *
****************************************
```

Police investigators are said to be able to collect ten wildly different reports of what happened in an accident from any ten bystanders. The Workshop on Software Testing and Test Documentation (Ft. Lauderdale, December 18-20) was not an accident, and I was not a bystander, but this report was requested after the Workshop ended--too late to listen more closely. Furthermore, my own bent is theoretical, so my testimony is suspect in any case. If you're still interested, Officer, put this on the blotter:

The Workshop was planned to provide maximum exposure to several testing viewpoints. (This meant minimum exposure to the Florida sun. I confess to sneaking to the beach at the expense of a couple papers, but it is a testimonial to the quality of the presentations that I was sorry afterward.) The Workshop was organized as a "single stream" with papers grouped under broad headings (Theoretical Aspects, Empirical Studies of Effectiveness, Test Documentation, Test Tools, Test Data Generation, Experience in Testing Large Systems, and New Approaches). I was not the only one who ducked out occasionally, but there was remarkably little of the session-cutting so common at parallel-stream conferences. (I mean cutting all sessions in parallel because the technical content is dilute to the point of zero concentration.) The single stream plan worked, and it deserves to be used more often. Credit for the design and implementation belongs to Ed Miller, who managed to retain his enthusiasm throughout, and worked tirelessly getting people together.

The usual dichotomy in program testing is between theory and practice. It is no longer so true that this divides people into academic and industry/government camps, because university researchers are beginning to be interested in problems of actual systems, and industry is acquiring more testing-research people. (Government is as usual left with paying the bill, without much control over the results.) The theory-practice split was evident at the Workshop, but exposure to the other side seemed more constructive than antagonistic. I think the theoreticians benefitted more, because they are looking for problems, while practicioners seek solutions. Here are some things I learned:

1) Not all theory comes from the usual sources. For example, a method of test-data generation was presented that solves some theoretical problems, and arises from a novel hardware-software analogy. The author was not only in industry, but primarily a manager.

2) Most of what is known about testing isn't being used, and this is slow to change. However deplorable the state of the art may be, those with access to testing tools find them far better than nothing. Yet nothing is what is available to most practicing programmers. For example, many compilers still lack source documentation aids like identifier cross referencing, and syntactic checking of consistency in subroutine linkage. Most runtime support systems lack debugging features like trace, monitor, and breakpoint. Anyone who has used such

features quickly agrees that they are valuable, but how valuable? Should a contractor or user be willing to pay substantial sums for them? Perhaps, but no one wants to be first out on the limb. And a study that might establish the value would be expensive and of doubtful validity... If these arguments apply to the ten-year-old compiler technology, what chance do less-established testing tools have of wide acceptance?

3) There is a hierarchy of problems in testing, ordered by how close we are to solutions. First, we have a fair understanding of unit testing of modules with clear specifications. (The essence of what seems to work is to rub the programmer's nose in the code until he or she discovers all the bugs therein. There are a number of computer-aided nose-rubbing systems that seem to work.) Second, at the level of integration testing we have little idea what to do, partly because specifications at this level are often poor, but also because the problem becomes too large for the expertise of one person, and nose-rubbing fails. Finally, there is the "large system" which comprises many integrated subsystems, and may be imbedded in a yet-larger mechanical and human system. Here the level of our understanding is reduced to building something, turning it on, and crossing our fingers. At this level, test documentation becomes very important, and if testing is in sad shape, documentation is worse.

4) Standing outside the hierarchy of 3) is the theory of testing, which isn't yet able to explain even the unit level. And there is no reason to believe that the problem hierarchy is causally connected in that if we solve problems at one level it will lead to success at the next. Without a sound theory we cannot say. The idea of "path testing" illustrates the situation admirably. There is no theoretical justification for this notion--a program can have every path tested and still contain errors. Yet path testing seems in practice to be a good nose-rubbing tool, successful at the unit level. At integration level the number of paths multiplies, and each loses significance. At the large-system level "paths" become "threads" through the functional modules to meet various requirements; the actual coverage of program paths is sparse. And to display the confusion in theoretical ranks, a theory session at the Workshop almost unanamously agreed that "path testing is worthwhile."

A special issue of the Transactions on Software Engineering is planned to include papers from the Workshop. It may be an injustice to say that few major new results were presented--I did miss the final 15 minutes. But even if there were more problems on the floor than solutions, I haven't enjoyed a conference for its technical content so much since back in the good old days of Theory of Computing, when all the papers were on context-free syntax, and it looked like we knew what we were doing.

Department of Computer Science
University of Maryland, College Park 20742

Richard Hamlet