TESTING THE LIMITS OF TEST TECHNOLOGY
Testing the Limits of Test Technology

It happens all the time in system-test laboratories: An attempt to recreate a bug logged during test results in an argument between developers who seldom admit faults and testers who often lack the necessary knowledge to prove or disprove the bug’s existence.

It is important that we deal with this conflict by improving test technology: Experience suggests that testing accounts for 25 to 45 percent of development cost. You may be as well acquainted with existing testing techniques — top-down, bottom-up, white-box, black-box, mutation, structural, functional, and so on — but your knowledge may be limited to methods that are applicable to conventional, sequential code.

Researchers in the forefront of testing are working on issues pertinent to distributed (unconventional, if you will) programs. Issues like complicated process and processor communications, event synchronizations, and network traffic contribute to the increased complexity of distributed testing.

Although we have adopted distributed architectures for most modern systems, we have no effective means to test the software executing on top of these architectures.

Furthermore, because modern distributed systems are expected to provide better and faster services, real-time requirements have become much more stringent. Real-time requirements add another dimension of difficulty to testing and debugging. In fact, process distribution and real-time requirements were the two major issues discussed at the first ACM SIGPlan and SIGOps Workshop on Parallel and Distributed Debugging in 1988.

Testing Characteristics

Effective testing improves quality and reduces costs. To conduct efficient tests, you must deal with both the execution mechanism and process control. The execution mechanism includes the software and hardware facilities that execute a test designed to meet quality criteria. Test-case selection, test-data generation, and test monitors are the indispensable elements of a test environment. Working with the test plan, the execution mechanism assists process control, which in turn drives the execution.

In the past, a great emphasis was put on the execution mechanism, focusing on test coverage. Now, statistical approaches are more popular and the emphasis has shifted to process control.

Process Control

Process control includes the management of requirements, staff, resources, schedules, acceptance criteria, and test results.

Staff management is essential, but seldom attracts academic research interest. One method that has considers staff is Cleanroom engineering, which uses specification, development, and certification teams, all carrying out different responsibilities.

To test large projects, you need rigorous process control to determine proper system partitions and to better synchronize the test phases. In the first article, "Improving Quality with a Manufacturing Process," Yezhak Levendel translates hardware manufacturing's process control to software by adopting the practice of first determining the error rate in each phase. Then, they applying statistical analysis to the results. This is a more down-to-earth approach to defect detection and repair than traditional approaches.

Many models use the number of cumulative defects as the index of quality in determining when to release software. Jason Su and Paul Ritter's article, "Experience in Testing the Motif Interface," details how one testing team developed release criteria and used tools to test software and ensure quality.

Fundamental testing theories are the foundation for testing methods. Examples are graph theory, formal languages, dataflow analysis, petri nets, statistical analysis, and stochastic process control. A good theoretical foundation not only formalizes the testing approach but also helps you develop more efficient test cases and data-generation methods and more interpretable results.


Execution Mechanism

Test-case selection and test-data generation cover the issues of test coverage, automatic testing, and test efficiency. Richard Denney has contributed a very stimulating article, "Test-Case Generation from Prolog-Based Specifications," that describes some ideas in generating tests with logic-based code. An article by Mary Jean Harrold and Mary Lou Sofia, "Selecting and Using Data for Integration Testing," is a comprehensive view of generating test cases for the interfaces of modules developed by different groups at different times.

Instrumentation tools determine the accuracy of the test results collected in the laboratory. Today, testers are trained to locate bugs quickly via various instrumentation and monitoring tools. The bugs must be corrected in no time and fixed applied to the products out in the field without causing interference to the normal operations. Noninterference monitoring is crucial to real-time testing to obtain a precise picture reflecting the true functionality and performance of the system under test. In their article, "Replay and Testing for Concurrent Programs," Richard Carver and Kuo-Chung Tai describe a way to monitor the testing of concurrent programs.

Immature Technology

Testing technology is not yet mature. Many testing techniques either do not scale up or rely on inflexible assumptions. For example, while symbolic-execution techniques have been presented in many academic papers, they are not yet used in industry.

Technology transfer in the testing area
is a slow and frustrating process. I strongly encourage the industry to make a sincere effort to upgrade its testing practices by incorporating modern techniques (automation, statistical control, and so on).

Meanwhile, industry should understand that many "innovative" testing ideas may not work in the real world. Pay particular attention to the underlying assumptions and limitations. For example, most C programmers would not feel restricted as to use of pointer arrays is reasonable.

On the other hand, academic researchers should validate their innovations through much larger trials. Understanding real-world practice will stimulate academics to solve more challenging problems, and understanding academic know-how will enhance industry's competitive edge. Mutual understanding is the key to successful technology transfer.

REFERENCES

Young-Fu Chang is a member of the technical staff at AT&T Bell Laboratories in Naperville, Ill. His research interests include specification verification, real-time systems, and performance analysis. Chang received a BS and MS in electrical engineering from National University of Taiwan and a PhD in computer science from the University of Illinois at Chicago. He is a member of the IEEE Computer Society.

Mikio Aoyama is a senior engineer at Fujitsu Limited. His research interests are real-time distributed software systems, software-development environments, and software-development management. Aoyama received a BS and MS in electronics from Okayama University. He is a member of the IEEE Computer Society, ACM, and the Information Processing Society of Japan. Address questions about this issue to Chang at AT&T Bell Labs, 650-365, 1200 East Warrenville Rd., Naperville, IL 60566, Internet: chang@alum.mit.com, or to Aoyama at Business Switching Software Division, Fujitsu Limited, 2-12-8-Shiro-kodanaka, Nakahara-ku, Kawasaki 211, Japan; mikio@mikio.nakahara.fujitsu.co.jp.

Software TestWorks
Apply the Power of Many Testing Experts!

Software TestWorks (STW) is an integrated system for software testing support and quality control function of the software life cycle. It includes management of software tests, regression testing, and coverage analysis. STW tools include:

**Coverage Analysis**

TCAT: Achieves branch level coverage analysis. TCAT allows you to determine what percentage of logical branch segments are actually being tested in any given test or sequence of tests, enabling you, the tester, to develop a higher degree of quality assurance.

S-TCAT: Measures test effectiveness by the percentage of function calls that are exercised in a program with a set of tests, relative to the total number of function calls that exist in the program system.

TCAT-PATH: Assists in analyzing source programs to determine the level of path coverage achieved in a given sequence of tests.

T-SCOPE: Gives the tester dynamic visualization of path digraphs and call-trees from test trace files. As a test trace executes, the tester can watch path coverage for module(s) selected. Also, works as a debugging tool, as the programmer can execute his code and watch path analysis in tandem.

**Regression Testing**

SMARTS: Software Maintenance and Regression Test System. Allows the tester to build test scripts to be executed automatically. Includes user-defined evaluations to determine PASS/FAIL for tests as part of a complete script language with provisions for conditional execution.

CAPBAK: A Keystroke and Mouse Capture/Replay System. Saves all delays, allowing for playback in real time, or the option of speeding up or slowing down. It can also record screens for use in regression testing at a later date. The program records in user-editable script files with conditional programmable playback.

EXDIFF: Expert Differencer. Allows differencing on ASCII files as well as graphic images, such as those recorded with CAPBAK. This tool will allow for masking to avoid differencing of selected parts of the file.

TOGEN: Test Data File Generator. Generates data according to a user-created template.

All of these tools are available separately or packaged together as part of Software TestWorks. Host platforms include 386/486 UNIX, Sun Workstations, HP/Apollo, IBM RS/6000, DECstations, IBM PC/XT/AT and others.

For more information, contact:

Software Research, Inc.
625 Third Street
San Francisco, CA 94107-1997
Phone: (415) 957-1441 or (800) 942.SOFT • FAX: (415) 957-0730

Reader Service Number 2