Software testing procedures

by LEE SPRAGUE
The International Bureau of Software Test, Inc.
Sunnyvale, California

ABSTRACT

Although perfect software does not exist, the use of thorough testing procedures greatly reduces the number of possible errors. The earlier testing procedures are introduced, the greater the developer's cost savings.

Independent software testing laboratories are working with standard-setting organizations to establish and define testing standards. Such laboratories provide objective, accurate, and economical testing through detailed testing plans and procedures that ensure that all items are tested and that tests can be repeated. Testing plans identify testing levels, types, methods, and procedures.

Levels of testing include unit, system, and integration. Types of testing include functional, destructive, regression, and documentation. Testing methods include procedural, outline, ad hoc, comparison, and compatibility. Software testing procedures consist of manual inputs, semiautomatic inputs, or automatic inputs and may require visual review of the output.

INTRODUCTION

No software is perfect, and typically it does not work as anticipated when first used. In fact, programs running well for years may conceal undiscovered bugs. The probability of error escalates as software becomes more complex or is integrated with other software. Although there is no perfect software, the use of thorough testing procedures greatly reduces the number of possible errors. In addition, the earlier these testing procedures are introduced, the greater the developer's cost savings.

WHY CAN'T SOFTWARE BE PERFECT?

Software errors occur whenever software does not work as anticipated. Because some computer programs are many thousands of lines long, humans find it difficult to construct anything that large and complex without an error. Even errors that seem small can have disastrous consequences. For example:

1. A misplaced character on a single punched card led a New England town to believe that its tax base was $7 million more than it actually was. Because the tax rate was set too low, the town ran short of money.
2. Miscalculations fed into the Bureau of Reclamation computers kept too much water behind dams, resulting in floods along the Colorado River.
3. The Vancouver stock index lost one point per day for more than a year because of a computer error. By the time the error was caught, the index had lost 574 points.
4. A United Airlines jet lost power and altitude on a flight into Denver. This may have been caused by an onboard computer that did not adjust to dropping outside temperatures, causing the engines to ice up and then overheat.
5. The British destroyer *Sheffield* was sunk during the Falkland Islands War when the ship's defense system recognized an incoming Exocet missile as a friendly weapon and failed to shoot it down.
6. The maiden flight of the U.S. space shuttle was delayed because of an error in one of the 500,000 instruction lines.

All these examples had significant financial implications. William Goss, President of the International Bureau of Software Test, Inc. has observed computer systems running successfully for years before a bug appeared. For example, an equation failed after years of use because of one unique number set that had never been provoked before.

Alan Borning, a computer scientist from the University of Washington, estimates that a Star Wars defense system might require ten million lines of software code. He said, "You can't even test it adequately. You can't tell the Russians, 'Fire off a couple of missiles—we want to test our program.' It has to work right the first time without being tested in any real way."[1]

In *Software Testing Techniques,* Boris Beizer has said that a 10-character input string has 2^{80} possible input streams and corresponding outputs and noted that complete functional testing in this sense is clearly impractical. At one test per microsecond, it would take approximately four times the current estimated age of the universe![2] Thus, it is difficult for software to be perfect.

WHAT IS THE STATUS OF SOFTWARE TESTING?

Until recently, software testing has been hit-or-miss, with few standards and little consistency. Most professional testing occurred in a few large organizations in which special departments tested company-developed programs. Independent authors and small software development companies depended on in-house programmers or friends to test software. However, it is extremely difficult for software developers to test their own programs effectively because they do not make mistakes when entering data into their own programs. They have difficulty imagining the questions and confusion that novice users might experience. Friends of software developers also do not have the impartiality necessary to provoke and uncover bugs in their friends' programs. Thus, a number of new software programs do not receive thorough and objective testing.

Independent software testing laboratories can provide objective, fast, accurate, and economical testing because their primary intent is to identify software discrepancies as early as possible. Early intervention decreases testing and rework costs.

WHY ARE SOFTWARE TESTING PLANS AND PROCEDURES USED?

Software testing plans and procedures ensure that software discrepancies are reasonably provoked. They also ensure that all items are tested and verified and that tests can be repeated. Without testing procedures, test engineers must act on their insights at a particular moment. With testing plans and procedures, test engineers can repeat the exact sequence or keystrokes to reconstruct tests.

WHAT DOES A SOFTWARE TEST PLAN INCLUDE?

The test plan defines the project's scope and contains the test's objectives, goals, and methods. It outlines the product's features and includes the test schedule with time frame and milestones when the various testing stages are to be completed. The test plan also identifies the test levels, test types, and test methods.

WHAT ARE THE LEVELS OF TESTING?

The level of testing is the degree of testing to be used for each product module. The three primary levels are unit, system, and integration; for every project, one or more of these testing levels is defined in the testing objective.

Unit Level

The unit level of testing requires isolation of each unit from all other units and identification and control of interfaces with all other units.

System Level

All features and functions of only one system are tested in concert. The interaction between the system's units and the interfaces between the units are tested.

Integration Level

Integration level testing involves testing in concert all features and functions of more than one system. This could involve testing the system's intended configuration as a whole and include software, firmware, and/or peripherals. If several applications are possible, the integration level may involve testing all features and functions in relation to each other. For example, if a word processing, spreadsheet, spelling checker, and scheduling package are integrated, they all must be tested together.

WHAT ARE THE FOUR MAIN TYPES OF TESTING?

The four main types of testing include functional, destructive, regression, and documentation.

Functional Testing

Test suites designed for functional testing represent normal use by the intended user. Functional testing may involve checking limits, passwords, menus and screens, error messages, and prompts. It may test for clear messages and organization. It may also check for acceptable response time, proper interfacing, and whether all configurations work. In addition, it may check different combinations of functions. Finally, it may test the accuracy of error-handling routines and the use of exit routines.

Destructive Testing

Destructive testing includes abnormal use and conditions by the intended user. Areas for destructive testing include values outside boundaries, insufficient memory and/or storage, insufficient file structure, and data overflow. Finally, destructive testing checks software recovery from simulated hardware errors and rejection of invalid entries or combinations.

Regression Testing

Regression testing is a repeat of functional and/or destructive tests to determine that reported discrepancies have been corrected and that no new discrepancies appear as a result of updates or system corrections.

Documentation Testing

Documentation testing verifies that the documentation and the system match and that the documentation is well organized, easy to understand, and technically and grammatically accurate. Specifically, documentation testing checks whether the manual answers user questions and whether it communicates clearly and consistently.

WHAT ARE THE METHODS OF TESTING?

Software testing methods include procedural, outline, ad hoc, comparison, and compatibility testing.

Procedural Testing Method

Procedural testing uses strict guidelines for each test path so that every feature and function the documentation describes is tested. Test procedures are specific steps, keystroke by keystroke, that reach the result the test case defines. Test procedures provide reusable, consistent, step-by-step guidelines for future testing.

Procedural testing has the advantage of providing detailed documentation to test all product areas so that testers can recreate problem areas. The disadvantages of procedural testing are that it is time-consuming; it does not allow for intuitive testing; and if areas are overlooked in the testing design, testing may not provoke discrepancies in these areas.

Outline Testing Method

Testing engineers use outline testing to guide them through the testing of interrelated parts of a system. All features listed in the outline are tested, but not necessarily in the order of appearance. Outline testing is partly an intuitive testing method, because the tester follows logical paths wherever they lead. Testers prepare test logs that provide a reference to the flow and logic of testing.

Ad Hoc Testing Method

Highly skilled testing engineers use ad hoc testing for specific areas. No documentation, other than software discrepancy reports, is produced; no guides are followed other than the testing engineer's intuition and the supplied documentation.

Comparison Testing Method

Comparison testing determines the similarities and differences between two or more systems. It may involve taking the reference section from the accompanying documentation, building a feature comparison matrix, and then testing each command on one system and then on the other system. This is followed by comparing the results and documenting any differences.

An alternative, if the systems under test are compatible, is to run an automated test simultaneously on each system, comparing the results and documenting any differences. If external hardware such as printers is involved, comparison testing can be accomplished by creating a master document that uses all possible features and functions and producing output on each printer. Then results are compared and differences are documented.

Compatibility Testing Method

Compatibility testing determines the ability of two or more systems to function interchangeably and identifies any functional differences. Areas of focus include keyboard input, CRT display, reproducible discrepancies, operating system functions, I/O functions, and disk exchange.

WHAT ARE SOFTWARE STANDARDS?

Software standards may be as formal as an established industry standard such as the "ANSI FORTRAN 77 Standard" or the information specifications to which the product was designed. The International Bureau of Software Test supports the ACM and the IEEE in establishing and defining software testing standards.

WHAT ARE TESTING PROCEDURES?

Testing procedures are detailed documents describing specific steps for accomplishing each test's goals. They include criteria for acceptance or rejection, as well as the configuration required to perform the test. Although the procedure's complexity varies with the testing level, the form of the procedures is similar. Software testing procedures require the test staff to review the entire software product for content, presentation, organization, and adherence to documentation. These procedures require manual inputs, visual review, or a combination of the two. Execution testing can consist of manual input, semiautomatic input, and/or automatic input.

WHAT ARE ADVANTAGES AND DISADVANTAGES OF VARIOUS TESTING PROCEDURES?

Manual Input

Test engineers manually enter data and control information as described in the test procedures. The primary advantage of manual input is the direct flow from the procedure's development to the test engineer for test execution. The main disadvantages are that (1) manual input of data introduces the possibility of an input error and (2) there must be continuous staffing for the total period of test execution and evaluation. This staffing requirement affects the testing cost because every time the tests are run, the investment in staff to run the tests remains the same.

Semiautomatic Input

Semiautomatic input consists of part manual procedures and part software test code execution. Primary advantages of semiautomatic test procedures are (1) the limited need for developing test code and (2) the reduced requirement for staff attendance and intervention. These advantages provide cost control. The main disadvantage of semiautomatic execution is that manual input of data introduces the possibility of an input error.

Automatic Input

The test engineer executes the test code, including the automatic test result verification, and reviews the results. The major advantages of automatic input are (1) the ability to repeat the test in exactly the same way every time, (2) the greatly reduced requirement for staffing during test execution so that the tester can start the test and review the results when the test is completed, and (3) lack of human error. The primary disadvantage of automatic input is the cost of investment in test code and test tool development before the test is executed. However, it must be recognized that this high cost is amortized over the number of times the test is performed.

WHEN SHOULD TESTING PROCEDURES BE INTRODUCED?

The single largest component of software cost is testing to discover software bugs. In fact, testing costs can exceed 50% of the software development expense. Therefore, software testing procedures should be introduced as early as possible in the development cycle. The earlier errors are found, the less expensive they are to correct. If testers review design specifications, they can call attention to problem areas and look for shortcomings. Testers can also be developing test plans while programmers are programming so that when coding is completed, testing can begin.

CONCLUSIONS

Although there is no perfect software, the use of thorough testing procedures greatly reduces the number of possible errors. In addition, the earlier testing procedures are introduced, the greater the resulting cost saving.

REFERENCES

1. Borning, A. "'The Last Bug' Computer Scientists Fear." *San Jose Mercury,* October 29, 1984, p. 1.
2. Beizer, B. *Software Testing Techniques.* New York: Van Nostrand Reinhold, 1983.