Software Testing

Judith S. Douglass, TAS, Inc.

Knowledge about software testing is critical not only for software designers, programmers, and professional testers, but also for buyers and users. Most of us fall into at least the last category!

Why should software be tested?

The goals of software testing, no matter who is doing the testing or where it is done, are to achieve confidence in the software and to reduce risk of errors caused by program "bugs." A bug is programming code that results in program function or output that is other than required program function or output. Bugs can be relatively innocuous (a typographical error on a screen), annoying (after a certain combination of keystrokes the program bombs), or very serious (the calculation procedure for averaging nutrient data is faulty).

Testing and debugging are not at all the same. Testing and test design focus on bug prevention and on uncovering bugs. Debugging may be the logical consequence of testing. The purpose of debugging is to find the code that led to program failure and to change the code accordingly.

There are ways that programmers can prevent or reduce program bugs. The source language can be very important. Programmers can adhere to certain stylistic criteria and/or design methodologies. Static analysis can be done. Software inspections can be performed. However, testing is needed even if these other features are incorporated.

Software testing can show some, but not all, defects in program code. Testing can demonstrate that the program function is correct or incorrect and can demonstrate that the performance is correct or incorrect. Testing can detect logic failure; this means that the program is doing what it was told to do, but that someone's reasoning (the developer or programmer) was faulty in some way.

Who should be part of the software testing process?

The buyer and/or user of software must play a major role in the software testing process. In contracted software development, personnel from the contracting institution should work with the developer during the planning and design process so that all parties agree on what the final system will consist of and how the system will look. Joel Gilman, in a Law Report column in the February 1991 *Systems Integration* (1) suggests that a test-criteria document should be drawn up if possible and included in the original system proposal or contract.

Buyers should consider software testing key features even for off-the-shelf software products. Bill Hancock, in a column titled "Confessions of a testing fanatic" in the November 12, 1992 *Digital News and Review* (2), related a horror story about an off-the-shelf word processor program crashing, with work lost prior to an important deadline. We've all come close to this at one time or another.

Programmers should be responsible for basic software testing. A programmer should never hand over a program to a tester or testing department without first processing enough test cases to determine whether the program is meeting specified requirements.

It should be noted, however, that testers and programmers have different goals when they are testing software. Boris Beizer, in *Software Testing Techniques* (3), an excellent basic book on testing, said that a tester is "one who writes and/or executes tests of software with the intention of demonstrating that the program does not work." He said that a programmer is "one whose tests (if any) are intended to show that the program does work."

What is software testing?

Software testing is a process, the purpose of which is to prevent and uncover "bugs" in the software and to determine whether systems meet specified requirements. It is by nature poorly defined, always incomplete, costly, and time-consuming. It has been estimated that testing consumes more than half the labor expended to produce a working program (3).

There are four general categories, often referred to as "stages," of software testing.

Unit testing is testing of the smallest testable piece of software (a single component of the system). The purposes of unit testing are to assess whether the unit satisfies its functional specification and/or whether its structure matches the intended design structure. Unit testing should be done by the programmer. There are a variety of tools a programmer can use to assess whether all paths the software can take have been tested.

In integration testing, the goal is to test what happens when units are combined. Usually if a problem is found, it has to do with information passed between units.

System testing, as the name implies, is aimed at testing the entire system. It tries to find problems other than those that can be attributed to units or interactions between units. In system testing, one might look at performance, security issues, and other issues of these types.

Acceptance testing is a broad category that provides final certification that the system is ready for real-world use. It may involve very intensive tests that look at the positive and negative aspects of the system. Functionality is closely examined. For contracted or in-house software, errors found at the acceptance testing stage usually involve some type of misunderstanding on the part of the developer or miscommunication between buyer and developer.

There are other terms for testing which may be done during the course of one or more of the testing stages. These include testing for reliability and usability, conformance to standards, interoperability, and regression testing.

Reliability testing assesses acceptability considering intents, actions, and decision processes of users. Measurement variables may include learning time, task performance time, error rates, error recovery time, and end-user satisfaction.

Conformance testing checks conformance to standards. Organizations issuing standards related to software include (but aren't limited to) the International Standards Organization, British Standards Institution, American National Standards Institute, and the Institute of Electrical and Electronics Engineers. However, there are other kinds of standards software can meet. For example, MS-DOS itself is a kind of a standard.

Interoperability testing assesses whether a system can work effectively with other systems meeting the same standards.

Regression testing involves tests created for a previous version of the software. When the system has undergone a change, testers usually repeat some or all of the tests performed on the last version of the software just to make sure that the change didn't adversely affect an unrelated function.

When should the testing process begin?

Testing, of course, can't begin until some code is written. However, the testing process should begin when the software development process begins. Time spent eliminating bugs is much shorter when testing is planned during the design phase than at the end.

System requirements should be written down and agreed upon by all parties, and these requirements specifications should be used as the basis of a testing plan. Software design objectives should include testability.

When should the developer stop testing and deliver the product? The decision may be based on metrics (measurements of error rates). However, the key decision will always be based on the resources available. If testing staff have nothing else to do, and lots of time until the software is needed, one might be able to be very cautious and extend testing at each stage for a long time. However, if the software was promised 6 months ago, and especially if a competitor is moving in, the developer might be willing to take the risk that the program works without serious bugs, even if he or she knows about some minor bugs.

It all boils down to risk assessment and being able to project the point when it is more-or-less safe to send out the product. If a faulty product is sent out, a new version can be released; however, this involves a whole new set of risk assessments.

The cost of sending out a new version must of course be considered. But a developer also has to consider the image of the company—sending out a new version can make a developer look responsive, innovative, or incompetent, depending on the timing and on the extent of change.

Where should software be tested?

This depends a lot on what kind of testing is planned. Presumably, the programmer will do some testing at his or her desk. If the company has testing facilities, or contracts out some of the testing, then testing will take place in these facilities. Sometimes a "pilot company" is created to simulate user conditions. And finally, some of the final acceptance testing should be performed at the buyer/user's facilities.

How should a testing plan be written and executed?

Testing plans at each stage should be based on structured requirements. These requirements should be written down and agreed upon by all parties before code is written. Structured requirements include any relevant definitions and input, process, and output requirements. In test planning, the steps are to finalize these requirements specifications, design the tests, map the tests to the requirements, and finally, designate the test cases and/or acquire a test set.

Test case design is its own science. The goals are to identify all of the types of cases that might occur under each scenario. After the code to be tested is written, the tester executes the planned tests, evaluates the results, and provides feedback. This feedback becomes a matter of record for the system. People who get the feedback would vary depending on the testing stage.

Examples of software tests

Example 1. Sample tests for food consumption analysis software

The TAS International Diet Research System® (TAS-DIET) uses U.S. Nationwide Food Consumption Survey (NFCS) and Continuing Survey of Food Intakes by Individuals (CSFII) data to report on food intake by the U.S. population and population subgroups. One of the definitions basic to the system is that the "3-day population" consists of respondents (for the survey in question) for whom three full days of survey data are available, regardless of whether food was actually consumed on any or all days. This definition applies to the total survey population, but can be extended to each survey subpopulation defined by the system; for example, the 3-day population of Hispanics consists of Hispanic respondents for whom three full days of survey data are available, regardless of whether food was actually consumed on any or all days.

In order to test whether the 3-day population definitions actually used by the system meet the definitions set out in the system requirements, analyses of 3-day average food consumption by the total population and each defined subpopulation should be performed and the results evaluated to determine N's reported by the system.

Running the analyses with the total population and each defined subpopulation might seem like an excessive number of test cases to test the definition of "3-day population" used by the system, but it really is possible that the system could be using the correct definition when using data for the total population but be totally off for one or more population subgroups.

To evaluate the results of these tests, the N's appearing on reports generated by the system should be checked against USDA documentation for N's in the total population and applicable subgroups. Because the documentation doesn't necessarily include N's for all of the population subgroups defined by the TAS system, the N's for the subgroups should be added together to check that the sums equal the N for the total population.

Example 2. Sample tests for food composition analysis software

A fictitious system, NUTRI-NOW, accepts an NDB number as input, retrieves appropriate nutrient data, and prints the nutrient data to the screen. The system must accept valid NDB numbers but not invalid numbers. There is only one processing requirement—to retrieve the most current USDA published nutrient data for the input NDB number.

For output, the system must display an error message if the input NDB number is invalid; for valid NDB numbers, the nutrient data must be shown in the appropriate order (which ordinarily would be included in the requirement specification) and in the appropriate units (which ordinarily would be specified for each nutrient).

In testing to see whether the system accepts valid NDB numbers, at least the lowest valid number and the highest valid number should be tested. If there are invalid numbers in the middle somewhere, the numbers before and after each of these numbers should be tested.

In testing to see whether the system rejects invalid NDB numbers, a wide variety of test cases should be processed. These test cases should include, at a minimum, the number lower than the lowest valid number; the number higher than the highest valid number; a number with spaces in the middle; a number with alpha characters in the middle; a "number" with alpha characters at the beginning; a number with special characters in it; and all zeros.

Summary

Testing should be more than a software life-cycle phase, and testing is not debugging. Testing should be carefully considered when designing software or planning software purchases. Software users should consider themselves software testers.

References

1) Gilman, J. (1991). Use test criteria to ensure client acceptance. *Systems Integration* 24 (2): 54.

2) Hancock, B. (1992). Confessions of a testing fanatic. *Digital News and Review* 9(21): 46.

3) Beizer, B. (1990). *Software Testing Techniques*, 2nd edition. New York: Van Nostrand Reinhold.