

Software Testing: Why & How

By Sherman Tan

20 Jul 07

Many people test software applications for various reasons. Some test software before making procurement, others test software to verify that a set of requirements are met, yet others test software to make the business users happy – this is usually the case when system integrators test software to satisfy their clients. Software testing often goes by a variety of names such as acceptance testing, usability testing, performance testing, robustness testing, reliability testing and many more.

Typically, in the software development cycle, the design phase takes the longest duration followed by the build phase and the least amount of time is allocated to the testing phase. At Microsoft, an enormous amount of time is spent on research and development for each new release of Windows™ and to supplement the testing efforts, Microsoft was the first to harness the general users in the beta testing of their softwares – this move has revolutionised the concept of beta testing.

Ironically, the building architect spent substantial amount of time to develop the ideal design with the builder given probably one quarter to a third of the entire project duration to construct the building. The home buyers who eventually pay hefty sum for the property does not even have the chance to “test” it. In Singapore, there have been recent property buyers who spent enormous sum of money buying properties without even visiting the show flat or looking at the building plan!

Let's forget about hot property purchases and come back to the basic question of why we perform software testing.

All Programs Contain Errors

It is a reality that most computers can perform only a limited series of simple actions but at very high speed – even billions of these simple actions in a second.

The ATM that we are familiar with that offer cash withdraw, balance enquiries, funds transfer and even perform bill payment has less than 100,000 lines of programming codes. The Air traffic control software that are used by air traffic controllers to guide planes to take off and landing has only 10 times more programming codes; ie just over 900,000 lines of programming codes.

However, many programs developed in recent years contain millions of lines of programming code. Microsoft Vista which was launched late last year was known to have some 50 million lines of programming codes which is 10 million more than Windows XP.

With so many lines of codes, errors will inevitably occur and very often it is impossible to eradicate them completely. Statistics showed that on average, commercially developed programs contain between 14 to 17 errors for every 1000 lines of codes. This means that Vista could potentially have more than 700,000 errors! Fortunately, most errors simply cause these programs to run slower (performance issue) or to perform unnecessary tasks. However, there are some errors that do cause inconveniences such as program freeze (needing reboot), lost data and even miscalculations.

Since many programs are increasingly being used in mission critical systems in various organisations, software testing is becoming more important than ever. However, not enough attention has been placed on software testing as this is always the last phase of the project cycle and when implementation date is often unrealistically set.

Before we take a look at how software testing was carried in the past, let's have a quick recap of what is a software program.

The Four Operations in the Information-Processing Cycle

Basically, a program is just a list of instructions that tells the computer how to perform the four essential operations to complete the task assigned. These four operations are:

- **Input:** Getting data into the computer
- **Processing:** Transforming data into information
- **Output:** Displaying information either on the computer screen or to the printers
- **Storage:** Holding programs and data for future retrieval and usage

The understanding of these four operations helps in our understanding of what needs to be tested and at which steps of information processing cycle that the tests have to be performed.

For instance, the roles of the testers is to ensure that the data for inputs are prepared in advance to solicit a set of pre-determined processing logic that would yield desirable/undesirable output. Even at the input stage, the testers must ensure that the program accept the input through validation and out-of-range checks.

In test planning, testers sometimes forget these four fundamental operations and focus on just getting the test scenario and test scripts developed.

There will be more discussions later on how to conduct input and output testing but now, onto the evolution of testing...

"Intuitive" Testing

In the late 1970s, software developers being a scarce and awed lot were given plenty of time to write their programs which they tested themselves. When the product was finally ready, the program leader got some end users to test it over a few days and sometimes over a weekend. If these users do not report any critical issues, the product went "live" on Monday.

During those times, software testing was seldom part of the system development lifecycle and testing is carried out in an unstructured manner. There was no test documentation to start with and knowledge and experience of the testers were also not part of the pre-requisite. Fortunately, this type of testing is rare nowadays.

Testing Using Design Documents

Recognising the limitation of unstructured testing, the next phase in the evolution of testing saw the use of design documents as the basis of conducting testing.

As a preparation, testers collected and reviewed technical and functional design documentation and used these documentations as the basis for designing the tests to be carried out. One of the setbacks of this sort of testing is that the testers followed the structure and scope of these functional and technical documents paying no attention to the testing priorities.

However, this method provided the test managers with a more structured form of carrying out the testing in comparison with the "intuitive" testing method used much earlier.

Requirement-Based Testing

As software become more complex and sophisticated to meet the demanding business environment and coupled with the fact that poorly developed software can have both financial and reputation risks, testing methodologies evolved over the years.

One of the improvements in testing methodologies saw the adoption of using requirements as the basis for testing. The rationale is the detailed requirements formed the specifications of the project and these detailed requirements provided the testers with better indications for preparing their test approaches and test cases.

In this testing methodology, testers used the list of detailed requirements to develop the test cases and execute them. In projects where there is a comprehensive list of detailed requirements before the testing planning cycle; there is always the mistaken belief that the software quality and reliability is a function of testing.

It is always the general belief that if you test an application extensively and found it to be working means the software is stable. Unfortunately, software testing has nothing to do with stability. The purpose of the testing is to ensure that the software application functions as it was designed to meet the requirements.

However, as in all projects, time and schedule for testing is always limited and in a number of projects, not all test cases can be developed for each and every of the detailed requirement. As a result, some of the requirements could not be tested. Hence, there is the need to look at other form of testing methodology to mitigate such risks.

Risk Based Requirement Testing

In many large and complex projects, not all requirements are known at the start of the project and in several cases detailed requirements may not be fully available at the start of testing. Such projects are characterized by the high number of concurrent activities leading to cross dependency of each work streams as well as the number of change requests. The need for a robust change management process is critical but is not the scope of this discussion.

Risk based requirement testing looked at those requirements that are important to the business. If some of these requirements are not fulfilled by the software; the risk and consequence to the business and operations as a whole will be significant.

In risk based requirement testing; the project sponsors assign priorities to the requirements so that testers can plan the test scenarios and develop the test cases to focus on these requirements with the highest priorities. In issue resolution, those requirements that have the highest priorities are also accorded the same level of priorities so that the software developers can determine which set of issues that they have to resolve first.

At the end of the testing, the project sponsors will be in the better position to determine the “quality” of the software application from the point of whether the requirements that have the most impact to business operations have been addressed. For those requirements that have lowest priorities, a decision can be taken whether these outstanding issues can be resolved later so that the software can go into production or to continue with the testing.

Input & Output testing

Earlier I mentioned that we will discuss more about Input and Output testing. For this discussion I refer to the book “How to Break Software – A Practical Guide to Testing” by James A Whittaker. In his book, the author provided several suggestions on how to conduct Input and Output testing.

For Input Testing, the following test scenarios are proposed:

- Apply inputs that force all error messages to occur
- Apply inputs that force the software to establish default values
- Explore allowable character sets and data types to invoke error handling capabilities
- Construct input lengths such that it causes overflow input buffers
- Find inputs that may interact and test combination of their values – this test condition is useful for cases where the software requires different data inputs and combination of inputs to commence a series of processing logic

- Repeat the same input or series of input numerous times that will consume resources and cause data initialization problem – this is relevant where back office function requires voluminous data inputs on a repetitive basis

In the case of Output Testing, the following test conditions could be considered:

- Force different outputs to be generated for each input – this will ensure that the testing uncovers all major behaviours associated with each input
- Force invalid outputs to be generated – this helps the testers to think of the special cases of input combinations that can lead to wrong answers being generated
- Force the properties of an output to change – this test those outputs that are editable, changeable or might otherwise cause internal data to differ between the initial rendering of the output and subsequent editing of the data
- Force the screen to refresh – this helps to find rendering problems in applications that are associated with screen output. This test is to create and modify objects on the screen so that the software forces the screen to refresh

The above test conditions should not be carried out in isolation for the purpose of “breaking” the software but rather to validate that functional requirements of the business users are met. These tests should also take into consideration the priorities of testing set by the project sponsors.

Conclusion

In all projects, time, budget and resource are always limited and in large and complex projects, it is always not possible to have a comprehensive set of requirements before the start of test planning to develop the test scenario and test cases.

Given these limitations, it is necessary to prioritise the budget; time and resources on the requirements that have the highest impact on the business operations if these requirements are not met and where work around are not possible. Assigning priorities to the requirements; testing them and resolving issues relating to these requirements in the same priority will ensure that those requirements that “Must Test” will precede those that “Should Test” and those that “Could Test”.

*The writer is the Principal Consultant & Director at Innovar Pte Ltd
(www.innovar.com.sg).*