International Journal of Computers & Technology
Volume 4 No. 1, Jan-Feb, 2013, ISSN 2277-3061

Study on Non Functional Software Testing

H.S Samra

Punjab Polytechnic College

dr_hssamra@yahoo.com

ABSTRACT

Improving software quality involves reducing the quantity of defects within the final product and identifying the remaining defects as early as possible. It involves both its functionality and its non-functional characteristics, such as usability, flexibility, performance, interoperability and security. In fact, defects found earlier in the development lifecycle cost dramatically less to repair than those found later. However, engineers cannot address non-functional quality requirements such as reliability, security, performance and usability early in the lifecycle using the same tools and processes that they use after coding and at later phases. Approaches such as stress testing for reliability, measuring performance and gauging user response to determine usability are inherently post-integration techniques. Accordingly, defects found with these tools are more disruptive and costly to fix.

Nonetheless, there has been a lop-sided emphasis in the functionality of the software, even though the functionality is not useful or usable without the necessary non-functional characteristics.

This research highlights the sporadic industry acceptance of some popular methods for designing for non-functional requirements and suggests some practical approaches that are applicable for companies that also must consider the demands of schedule and cost.

INTRODUCTION

Non Functional testing is the testing of a software application for its non-functional requirements. The names of many non-functional tests are often used interchangeably because of the overlap in scope between various non-functional requirements. For example, software performance is a broad term that includes many specific requirements like reliability and scalability.

Special methods exist to test non-functional aspects of software. In contrast to functional testing, which establishes the correct operation of the software (for example that it matches the expected behavior defined in the design requirements), non-functional testing verifies that the software functions properly even when it receives invalid or unexpected inputs. Software fault injection, in the form of fuzzing, is an example of non-functional testing. Non-functional testing, especially for software, is designed to establish whether the device under test can tolerate invalid or unexpected inputs, thereby establishing the robustness of input validation routines as well as error-management routines. Various commercial non-functional testing tools are linked from the software fault injection page; there are also numerous open-source and free software tools available that perform non-functional testing.

NON FUNCTIONAL CATEGORIES

Performance Testing

Performance is one of the most important aspects concerned with the quality of software. It indicates how well a software system or component meets its requirements for timeliness. Performance testing is in general executed to determine how a system or sub-system performs in terms of responsiveness and stability under a particular workload. It can also serve to investigate measure, validate or verify other quality attributes of the system, such as scalability, reliability and resource usage.

Load Testing

Load testing is primarily concerned with testing that the system can continue to operate under a specific load, whether that be large quantities of data or a large number of users. This is generally referred to as software scalability. The related load testing activity of when performed as a non-functional activity is often referred to as endurance testing.

Load testing is the process of putting demand on a system or device and measuring its response. Examples

– simulating multiple users accessing a web server

– giving a huge document to a word processor

– subjecting a mail server to a large amount of e-mail traffic

– writing and reading data to and from a hard disk continuously.

It helps to identify the maximum operating capacity of an application as well as any bottlenecks and determine which element is causing degradation

Volume Testing

Volume testing refers to testing a software application or the product with a certain amount of data. E.g., if we want to volume test our application with a specific database size, we need to expand our database to that size and then test the application's performance on it.Volume testing is a way to test functionality. The purpose of volume testing is to determine system performance with increasing volumes of data in the database.

Volume testing is done against the efficiency of the application. Huge amount of data is processed through the application (which is being tested) in order to check the extreme limitations of the system.

Volume Testing, as its name implies, is testing that purposely subjects a system (both hardware and software) to a series of tests where the volume of data

International Journal of Computers & Technology
Volume 4 No. 1, Jan-Feb, 2013, ISSN 2277-3061

being processed is the subject of the test. Such systems can be transactions processing systems capturing real time sales or could be database updates and or data retrieval.

Volume testing will seek to verify the physical and logical limits to a system's capacity and ascertain whether such limits are acceptable to meet the projected capacity of the organization's business processing.

Stress Testing

Stress testing is a way to test reliability under unexpected or rare workloads. stress testing refers to tests that determine the robustness of software by testing beyond the limits of normal operation. Stress testing is particularly important for "mission critical" software, but is used for all types of software. Stress tests commonly put a greater emphasis on robustness, availability, and error handling under a heavy load, than on what would be considered correct behavior under normal circumstances.

Testing used to determine the stability of a system

• Testing beyond normal operational capacity, often to a

breaking point, in order to observe the results.

• The goals may be to ensure the software does not crash in conditions of insufficient computational resources (such as memory or disk space).

Stability Testing

Stability testing (often referred to as load or endurance testing) checks to see if the software can continuously function well in or above an acceptable period.

There is little agreement on what the specific goals of performance testing are. The terms load testing, performance testing, reliability testing, and volume testing, are often used interchangeably. In a stability test, a component is pushed to the point of crashing to learn where its limitations are. Stability testing can also be used to determine how long a component can operate under high stress, and at what point errors other than a total crash start to occur. A number of computer programs are designed to be used in stability testing, with people loading the software onto their computer and allowing it to run the testing, and people can also conduct tests manually.

Stability testing is a very important part of product development. It is used to determine the limitations of a product before it is released, and to identify areas which may need improvement or modification before product release. The stability test is also part of the quality assurance testing used to show stockholders the capabilities of the product, and to assure them that the product is being meticulously tested before it goes to market.

Usability Testing

Usability testing is a technique for ensuring that the intended users of a system can carry out the intended tasks efficiently, effectively and satisfactorily. Usability testing can be carried out at various stages of the design process. In the early stages, however, techniques such as walkthroughs are often more appropriate.

It is carried out in order to find out if there is any change needs to be carried out in the developed system (may it be design or any specific procedural or programmatic change) in order to make it more and more user-friendly so that the intended/end user who is ultimately going to buy and use it receives the system which he can understand and use it with utmost ease. Any changes suggested by the tester at the time of testing, are the most crucial points that can change the stand of the system in intended/end user's view. Developer/designer of the system need to incorporate the feedback (here feedback can be a very simple change in look and feel or any complex change in the logic and functionality of the system) of usability testing into the design and developed code of the system (the word system may be a single object or an entire package consisting more than one objects) in order to make system more and more presentable to the intended/end user. Developer often try to make the system as good looking as possible and also tries to fit the required functionality, in this endeavor he may have forgotten some error prone conditions which are uncovered only when the end user is using the system in real time. Usability testing helps developer in studying the practical situations where the system will be used in real time. Developer also gets to know the areas that are error prone and the area of improvement.

Usability Evolution Method

Usability test, as mentioned above is an in-house dummy release before the actual release of the system to the intended/end user. Hence, a setup is required in which developer and testers try to replicate situations as realistic as possible to project the real time usage of the system. The testers try to use the system in exactly the same manner that any end user can/will do. Please note that, in this type of testing also, all the standard instruction of testing are followed to make it sure that the testing is done in all the directions such as functional testing, system integration testing, unit testing etc.

The outcome/feedback is noted down based on observations of how the user is using the system and what are all the possible ways that also may come into picture, and also based on the behavior of the system and how easy/hard it is for the user to operate/use the system. User is also asked for his/her feedback based on what he/she thinks should be changed to improve the user interaction between the system and the end user. Usability testing measures various aspects such as:

* How much time the tester/user and system took to complete basic flow?
* How much time people take to understand the system (per object) and how many mistakes they make while performing any process/flow of operation?
* How fast the user becomes familiar with the system and how fast he/she can recall the system's functions?
* And the most important: how people feel when they are using the system.

Compatibility Testing

Compatibility testing is a type of testing used to ensure compatibility of the system/application/website built with various other objects such as other web browsers,

hardware platforms, users (in case if its very specific type of requirement, such as a user who speaks and can read only a particular language), operating systems etc. This type of testing helps find out how well a system performs in a particular environment that includes hardware, network, operating system and other software etc. as shown in figure 1.

Developers generally lookout for the evaluation of following elements in a computing environment (environment in which the newly developed system/application is tested and which has similar configuration as the actual environment in which the system/application is supposed to fit and start working).

Hardware: Evaluation of the performance of system/application/website on a certain hardware platform. For example: If an all-platform compatible game is developed and is being tested for hardware compatibility, the developer may choose to test it for various combinations of chipsets (such as Intel, Macintosh), motherboards etc.

Browser: Evaluation of the performance of system/website/application on a certain type of browser. For example: A website is tested for compatibility with browsers like Internet Explorer, Firefox etc. (usually browser compatibility testing is also looked at as a user experience testing, as it is related to user's experience of the application/website, while using it on different browsers).

Network: Evaluation of the performance of system/application/website on network with varying parameters such as bandwidth, variance in capacity and operating speed of underlying hardware etc., which is set up to replicate the actual operating environment.

Peripherals: Evaluation of the performance of system/application in connection with various systems/peripheral devices connected directly or via network. For example: printers, fax machines, telephone lines etc.

Compatibility between versions: Evaluation of the performance of system/application in connection with its own predecessor/successor versions (backward and forward compatibility). For example: Windows 98 was developed with backward compatibility for Windows 95 etc.

Softwares: Evaluation of the performance of system/application in connection with other softwares. For example: Software compatibility with operating tools for network, web servers, messaging tools etc.

Operating System: Evaluation of the performance of system/application in connection with the underlying operating system on which it will be used.

Databases: Many applications/systems operate on databases. Database compatibility testing is used to evaluate an application/system's performance in connection to the database it will interact with.

Figure 1: Compatibility Matrix

Endurance Testing

It is also known as Soak testing. Endurance testing involves testing a system with a significant load extended over a significant period of time, to discover how the system behaves under sustained use. For example, in software testing, a system may behave exactly as expected when tested for 1 hour but when the same system is tested for 3 hours, problems such as memory leaks cause the system to fail or behave randomly.

The goal is to discover how the system behaves under sustained use. That is, to ensure that the throughput and/or response times after some long period of sustained activity are as good as or better than at the beginning of the test. It is basically used to check the memory leaks. Endurance testing involves examining a system while it withstands a huge load for a long period of time, and measuring the system's reaction parameters under such conditions. Performance quality may also be tested to make sure that both the result and the reaction times - after a defined long period of continuous load - are degraded no more than a certain specified percentage from their values at the beginning of the test. For instance, in program testing, a system may perform exactly as anticipated when tested for one day. However, when it is tested for three days, hardware resource issues, such as a memory shortage, can cause the system to crash or function improperly.

In the field of software, endurance testing may involve testing the operating system and the computer hardware up to or above their maximum ratings for a long period of time. Some companies may endurance test a software package for up to a year, while also applying external loads such as Internet traffic or user actions. During endurance tests, memory consumption is observed to determine potential failures. Performance quality is sometimes also monitored during endurance testing.

Security Testing

The security testing is performed to check whether there is any information leakage in the sense by encrypting the application or using wide range of software's and hardware's and firewall etc. The six basic security concepts that need to be covered by security testing are: confidentiality, integrity, authentication, availability, authorization and non-repudiation.

International Journal of Computers & Technology
Volume 4 No. 1, Jan-Feb, 2013, ISSN 2277-3061

•Authentication - Testing the authentication schema means understanding how the authentication process works and using that information to circumvent the authentication mechanism. Basically, it allows a receiver to have confidence that information it receives originated from a specific known source.

•Authorization - Determining that a requester is allowed to receive a service or perform an operation.

•Confidentiality - A security measure which protects the disclosure of data or information to parties other than the intended.

•Integrity – Whether the intended receiver receives the information or data which is not altered in transmission.

•Non-repudiation - Interchange of authentication information with some form of provable time stamp e.g. with session id etc.

CONCLUSION

Clearly, software doesn't have to be 100% bug free. In fact, one of the hardest problems with testing is to know when to stop. If your company puts a team of testers on a project, and they spend four weeks on the finished product, they may find a lot of bugs the first week, some the second week, few the third week, and none the fourth week. But just because they found no bugs in the fourth week doesn't mean there are none. There is no practical way to prove that any piece of real world software is devoid of bugs, even a well-tested piece of software. In addition, functionality for expert users of software often doesn't get tested as well as the basic functionality, because testers are rarely expert users. No one wants to get a reputation for software that is not robust in the eyes of their expert users, because expert users have an impact on the usage habits of novice users. If these users get upset, your entire user base could slowly migrate to another product, even if you tested it fairly thoroughly!

Testing is generally considered costly and a nuisance. But as we have just seen, it is a necessary nuisance. The goal for most companies should be to do the best job testing possible and to minimize the costs. The idea that seems to work best is "test early and test often." Robustness isn't a module that can be bolted onto the side of a preexisting system -- it is far more cost-effective to develop robust software if you strive for this quality from day one. Similarly, the more software is tested, the more bugs will be found (although bad test strategies are often ineffective ones).

Testing can show the presence of faults in a system; it cannot prove there are no remaining faults

REFERENCES

[1] Myers, Glenford J. (1979). The Art of Software Testing. John Wiley and Sons. ISBN 0-471-04328-1..

[2] Srinivasan Desikan, Gopalaswamy Ramesh Software Testing: Principle & Practise. Pearson Education.

[3] Andreas Spillner, Tilo Linz, Hans Schaefer Software Testing Foundations Shoff publisher & distributor

[4] Exploratory Testing, Cem Kaner, Florida Institute of Technology, Quality Assurance Institute Worldwide Annual Software Testing Conference, Orlando, FL, November 2006.

[5] Wasif Afzal *, Richard Torkar, Robert Feld A systematic review of search-based testing for non-functional system properties . Information and software Technology Journal.

[6] J. Nielsen, "Heuristic Evaluation". In Jakob Nielsen and Robert L. Mack, editors, "Usability Inspection Methods". John Wiley and Sons, Inc.

1994.

[7] Mauro, C.L.: Professional usability testing and return on investment as it applies to user interface design for web-based products and services.

[8] Y. Lei, K.-C. Tai, In-parameter-order: A test generation strategy for pairwise testing, in: HASE'98: The Third IEEE International Symposium on High Assurance Systems Engineering, IEEE Computer Society, Washington, DC, USA, 1998, pp. 254–261.

[9] P. McMinn, Search-based software test data generation: a survey, Software Testing, Verification and Reliability 14 (2) (2004) 105–156