Requirements Based Testing of Software

Tabinda Sarwar, Wajiha Habib, Fahim Arif
National University of Sciences and Technology, Islamabad
Pakistan

***Abstract*—Testing is the method that assures that the developed system conforms to the specification and no error arises during system usage. No system is deployed without testing as testing produces confidence in the system. This paper presents the Requirements Based Testing (RBT) technique for testing the Usability and Functional Requirements of a software. The main emphasis of this paper is to test the usability of a software, as being a non-functional requirement that is perceived subjectively it is difficult to measure it. A generic template has been proposed for testing usability. For the purpose of evaluation, this method is applied to test a software named "Programming Teaching Aid", a system that helps students in understanding the programming language C++.**

Keywords*—*Software Testing, Requirements Based Testing, Requirements, Usability, Non-functional Requirements

I. INTRODUCTION

Interactive system involves extensive user involvement which includes both the end user and the developers [1]. Testing is done to access the quality of software and to ensure that no hindrance is caused during the interaction of user with the software. Different approaches to testing have been adopted e.g. acceptance testing, white box testing, black box testing, alpha testing, beta testing, unit testing, integration, system testing, functional testing and many other. It is impossible to specify one best testing technique that is applicable for testing various types of software available nowadays. Usually more than one approach is used for testing a software system.

While dealing with software systems it is important to identify the interactive properties of the system. An interactive property is a feature of a software which is the subject of analysis and evaluation [10].

Usability is a major characteristic of a software system. Usability is defined as "The capability of the software product to be understood, learned, used and attractive to the user, when used under specified conditions" [18].

Usability problems have caused some serious damages over the years. Usability problems in medical devices are of special consideration as they can be life threatening. Example of such medical devices with usability problems in their underlying software system is infusion pumps. These infusion pumps parse numbers in several ways depending on their mode, and thus provide an unpredictable experience for the user, particularly in syntax error handling [11] potentially with the result that patients are given incorrect drug doses [12] and these problems are not limited to specialist devices [12].

The proposed requirement based test approach helps release software in time, with required features and lesser usability problems hence reducing rework and minimizing potential damages. Requirements based test demonstrates that the software meets the requirements by deriving test cases that are developed in accordance with the requirements [2]. Requirements-based tests are "black box" (or functional) tests. This process addresses two major issues: first, verify that requirements are testable and second, designing a necessary and sufficient (from a black box perspective) set of test cases from those requirements [3].

This paper is organized into different Sections. Section II presents the work related to RBT that has been done up till now. Proposed method is presented in Section III. Section IV presents the implementation of proposed idea on an interactive system "Programming Teaching Aid". Finally Section V presents the conclusion of this work.

II. RELATED WORK

A large portion of work has been done for testing the functional and non-functional requirements of the system. Common techniques of testing involves black-box testing, white-box testing, stress testing, performance testing, component testing, interface testing, equivalence partitioning, structural testing and acceptance testing [13].

A function testing method based on data flow graph for testing the software interface and inner performance [4]. For testing a specified function, the data flow graph is derived in accordance with the logic completion process and interface and then all the paths in the data flow graph are executed to test the function comprehensively.

An integrated approach to cover testing of interactive systems has also been proposed, incorporating modeling of system behavior with fault modeling and minimizing the test sets for the coverage of "Event Sequence Graphs (ESG)" and Event Sequences (ES) that represent the human

ISBN: 978-1-4673-5256-7/13/$31.00 ©2013 IEEE

-computer interaction [7].

In another approach for testing the software, the user interface prototypes are developed iteratively with potential users [5]. Modeling and testing of interactive systems with a state-based model was formulated in [8]. Another state-oriented group of approaches to test case generation and coverage assessment is based on model checking [9].

Another technique is generating test cases for interactive systems by using data flow graph [6]. For testing a specified function, the data flow graph is derived first according to their logic completion process and interface. After this, test paths and test data could be derived based on the data flow graph.

Usability is the core concern of interactive systems. According to [18] measurement of system usability consists of three usability attributes:

1. Effectiveness: How well do the users achieve their goals using the system? [14]
2. Efficiency: What resources are consumed in order to achieve their goals? [14]
3. Satisfaction: How do the users feel about their use of the system? [14]

To provide the operational measurement of usability several attempts have been made in the domain of human computer interaction (HCI). According to [15] five main factors that define usability are: learnability, efficiency, memorability, errors, and satisfaction.

A layered model of usability [16], shown in Figure 1, helps in developing better understanding of usability and factors that affect it.

III. METHOD OVERVIEW

The method devised for performing RBT on an interactive system is shown in Figure 2.

a) Checking for Testable Requirements

In this paper we have assumed that that requirements gathering process is complete and the requirements have been transformed into code that can be tested. A general principle of requirements engineering is that requirements should be testable [13]. So to verify that the requirement is testable, a generic template is designed, shown in Table 1. If the requirement can be transformed into the template, mentioned in Table 1, than it is testable.

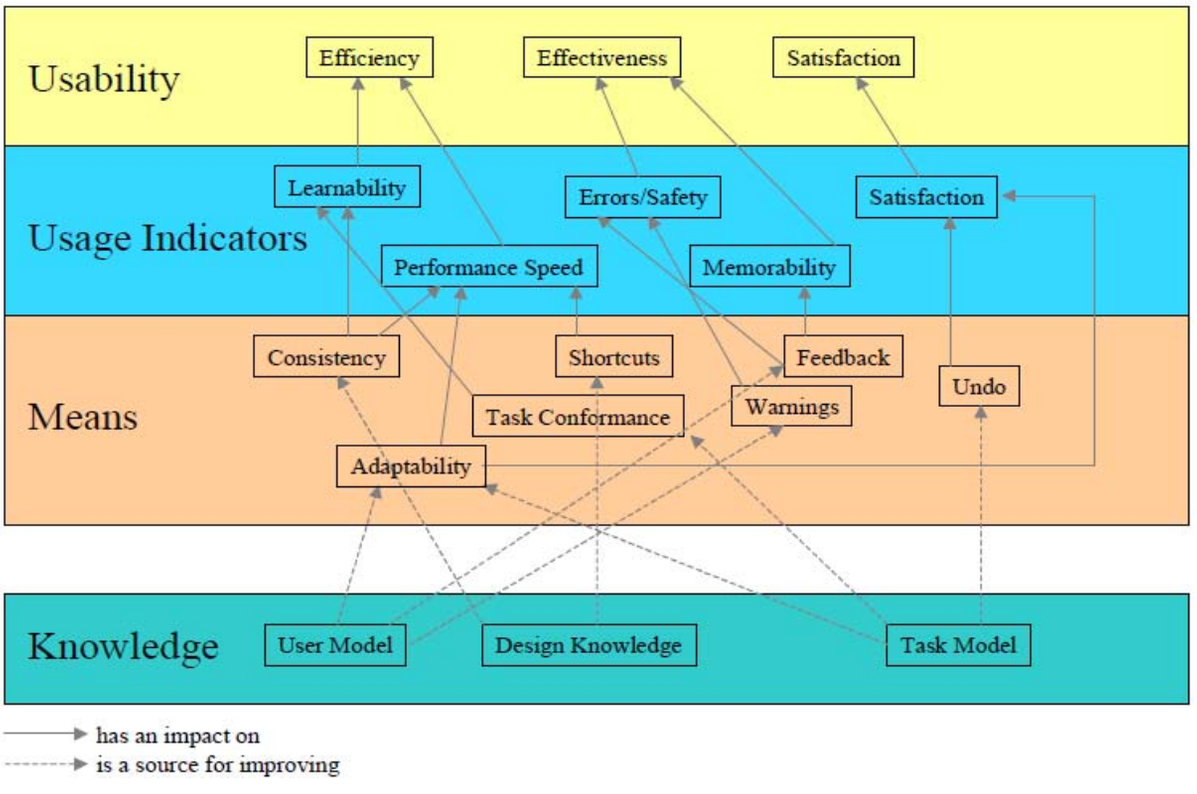

Fig 1: Layered Model of Usability [16]

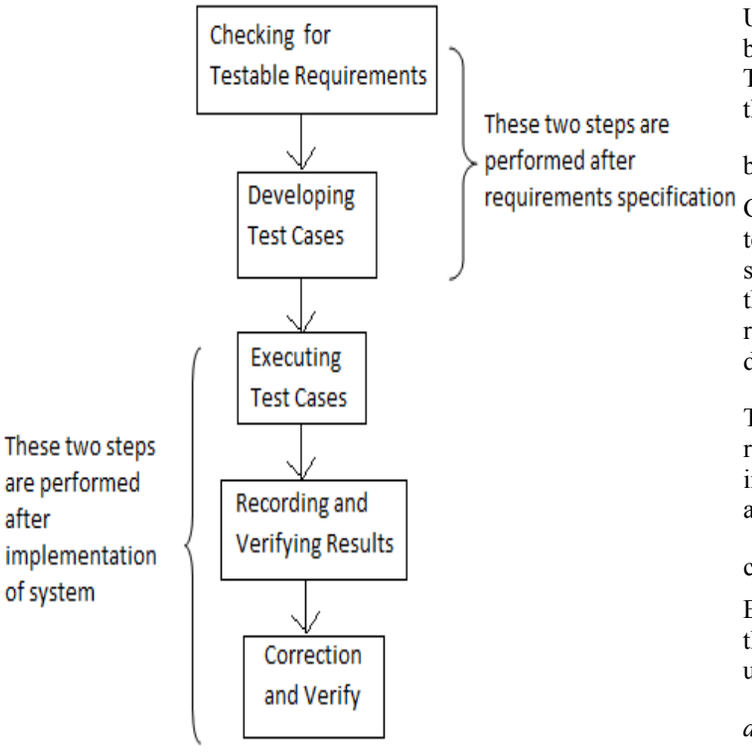

Fig 2: Process for Testing Interactive System

Table 1: Generic Template for Verifying Testable Functional Requirements

Category	Description (why it is needed?)	Test Environment	Input	Output
Function (High- Level View)	Description of function	Developer side/ User side	Input for the function	Output of the function

Usability is a non-functional requirement that is described by user subjectively. So to measure usability objectively, Table 2 describes the usability categories and their measures that will help in developing and executing the test cases.

b) *Developing Test Cases*

Generic template for test cases is shown in Table 3. For testing the usability of the system, the generic test case is shown in Table 4. This generic test case can also be used by the requirements engineers to gather the usability requirements (in the form of sub-categories), which is a difficult task otherwise.

This generic test case can help in testing the non-functional requirement (usability) of a wide range of software that includes desktop application, web-application, mobile application and real time systems.

c) *Executing Test Cases*

Execute the test cases designed using Table 1 (for testing the functional requirements) and Table 3 (for testing the usability).

d) Record and Verify Results

Record the actual results obtained after executing the test cases and compare them with the expected results. Deviation from the expected result means error/fault which has to be eliminated.

e) Correction and Verification

Find the source of error/fault (if deviation from the expected result occurs) and correct it. Again execute the test cases to verify the correctness.

Table 2: Criteria for Objective Measure of Usability

Category	Pre-Condition	Execution	Measure
Effectiveness	Requirements grouped in the form of tasks	Execute each task separately	Successful or unsuccessful execution of task
			Error handled or not handled that occurred in task execution
Efficiency	The software goals grouped in the form of tasks	Execute each task separately	Time taken to complete task
			Effort in the form of mouse and keyboard clicks
Satisfaction	Execution of successful tasks	--	User recommendation of software to others in form of yes/no
			User likeness of software compared to others in form of yes/no
Learnability	Execution of successful tasks	--	Time spent on using help materials like support or manuals
			Number of tasks remembered by user during execution of task

Table 3: Generic Template for Test Cases

Test Case Number	Pre-Condition	Input	Dependency	Expected Output
Number of test case	Condition before the input	Input that leads to success	Dependency on other function (this helps in finding error)	Successful output

ISBN: 978-1-4673-5256-7/13/$31.00 ©2013 IEEE

Table 4: Generic Test Case for Usability

Category	Category Description	Intention	Expected Result	Actual Result
Effectiveness	To find the success of task and failure handling	The number of times the task was successful	Ratio of expected number of task accomplished/total tasks	Ratio of actual number of task accomplished/total tasks
		Number of times the failure was handled by the system	Ratio of expected number of failures handled /total errors occurred	Ratio of actual number of failures handled /total errors occurred
Efficiency	The accuracy and completeness of goals achieved in relation to resources [17]	Time taken by user to complete one task	Expected time in minutes or seconds	Actual time in minutes or seconds
		Number of key press and mouse clicks required to do one task	Expected number of key press and mouse clicks	Actual number of key press and mouse clicks
Satisfaction	To find the user satisfaction with the system	Per cent of customers who would recommend it to a friend	Expected percentage of customers	Actual percentage of customers
		Per cent of customers that rate the product as "more satisfying" than a previous product (if exist)	Expected percentage of customers	Actual percentage of customers
Learnability	To find the learning time	Percent time spent using manual [14]	Expected percentage of time	Actual percentage of time
		Number of steps remembered after task completion	Expected number of steps	Actual number of steps

IV. CASE STUDY

This methodology was applied for testing software, Programming Teaching Aid (PTA) that takes a C++ code for input and generates the code description along with flowchart depicting the code sequence.

a. Checking Requirements' Testability

First verify that the requirements are testable or not using Table 1. For the sake of simplicity we have mentioned only 3 functional requirements in Table 5.

b. Testing Functional Requirements and Usability

Due to the limitation of space, 4 test cases are depicted in Table 6 for functional testing. The usability testing is shown in Table 7.

c. Correction and Verification

The identified deviations were handled by correcting the underlying code. The test cases were re-executed to verify the correctness of code. Few errors that were corrected are:

- The handout option was not operating.
- The code line with spaces at the start was skipped in the description.
- Manual Button was not working.
- Flowchart was not shown for every code.
- Description of "Using" keyword was not shown
- "\n", "\t" etc literals were not shown in the description.
- Description of "endl" was not shown.

V. CONCLUSION

This paper does not include the detail of how to get good requirements. It takes this thing as a premise that requirements are well written and discusses how to generate test cases from those well written requirements for an interactive system. Since usability is the core concern in case of software so it remained the main focus of this paper, but testing of functional requirements was also discussed.

The approach presented in this paper tests usability in a quantitative way which makes it easy for the development organization to assess how much a particular system is usable, moreover potential flaws in the system can also be detected easily which serves as a of basis improving the system. The test cases mentioned for usability can be used for other types of system like real-time systems as well.

Table 5: Checking the testability of requirements

Category	Description	Test Environment	Pre-Condition	Input	Expected Output
Testing Code – Description functionality	To test the correct generation of description of C++ code	Developer side	N/A	C++ programming Code	Correct description of each statement
Testing Flow chart generation of code	To test the generation of flowchart from code description	Developer side	C++ programming code entered into the software	Description of the code	Flow chart generated from description
Generate Handouts of the description generated	To test that the handouts of the description are in correct, complete, readable and formatted form	Developer side	C++ programming code entered into the software	Description of the code	Handouts generated from the description

Table 6: Test cases for testing the functional requirements of PTA

Tester	***			**Date**	***
Software	Programming Teaching Aid				
Pre-Condition	Software in operating normally				
Test Case Number	**Pre-Condition**	**Input**	**Dependency**	**Expected Output**	**Actual Output**
T2	N/A	Sample Code of "Hello World" (S1)	N/A	Correct description of each code's statement	Description of the code line containing white space in the start is not shown
T3	N/A	Sample code for testing if-else statement(S2)	N/A	Correct description of each code's statement	Correct description of each statement
T17	Sample code (S1)	Description of each statement of code	"Code-Description" functionality	Generation of flowchart	Flowchart not created
T20	Sample code (S1)	Description of each statement of code	"Code-Description" and "Generation of Flowchart" functionality	Generate the handout confirming to standard "Handout-Template"	"Unhandled exception" error message.

Table 7: Testing the Usability of PTA

Tester	***		**Date**	***
Software	Programming Teaching Aid			
Pre-Condition	Software in operating normally			
Category	**Category Description**	**Intention**	**Expected Result**	**Actual Result**
Effectiveness	To find the success of task and failure handling on code-to-description generation	The number of times code generation was successful	14/15	12/15
		Number of times system handled failures occurred during description generation successfully	14/15	9/15
Efficiency	The accuracy and completeness of code-to-description achieved in relation to resources	Time taken to generate description of one code	Not more than 1 minute	1 minute
		Number of mouse clicks required to generate description of one code	1 click	1 click
Satisfaction	To find the user satisfaction with the PTA's code-to-description functionality	Percent of positive to negative adjectives used by the user to describe product	More than 75 %	80 %
		Percent of users who would recommend it to a friend	More than 70 %	72 %
		Per cent of customers that rate the product as "more satisfying" than a previous product (if exist)	More than 70 %	69 %
Learnability	To find the learning time of PTA	Percent time spent using manual	Less than 35 %	30 %
		Number of steps remembered after task completion	At least 4	Average 3

REFERENCES

1. J. Grudin, “Interactive Systems: Bridging the Gap between the Developers and Users”, IEEE Computer, Volume 24 Issue 4, April 1991.
2. J. Ryser, S. Berner and M. Glinz, “On the State of Art in Requirements Based Validation and Test”, in ACM Digital Library, March 1998.
3. Bender RBT Inc., “Requirements Based Testing Process Overview”, 2009.
4. C. Wen-Jing, X. Sheng-Hong, and Y. Xiu-Xia, “A Function Testing Method for Interactive Software”, IEEE Second International Workshop on Computer Science and Engineering, 2009.
5. J. Anderson, F. Fleek, K. Garrity, and F. Drake, “Integrating Usability Techniques into Software Development”, IEEE Software January/February 2001.
6. C. Wen-Jing and X. Sheng-Hong, “On Test Case Generation for Interactive Software”, International Conference on Computational Intelligence and Software Engineering CiSE, 2009.
7. F. Belli and C. J. Budnik, “Minimal Spanning Set for Coverage Testing of Interactive Systems”, Springer-Verlag Berlin Heidelberg 2005
8. J. Offutt, L. Shaoying, A. Abdurazik, and P. Ammann, “Generating Test Data From State-Based Specifications”, The Journal of Software Testing, Verification and Reliability, Volume 13 Issue 1, pp.25-53, Medgeh 2003.
9. A. Gargantini, C. Heitmeyer, “Using Model Checking to Generate Tests from Requirements Specification”, Proc. ESEC/FSE ’99, ACM SIGSOFT, pp. 146-162, 1999.
10. G. D. Abowd, J. Coutaz and L. Nigay, “Structuring the Space of Interactive System Properties”, Proceedings of the IFIP TC2/WG2.7 Working Conference on Engineering for Human-Computer Interaction, 1992.
11. H. Thimbleby and P. Cairns, “Reducing number entry errors: Solving a widespread, serious problem,” Journal of The Royal Society Interface, Volume 51 Issue 7, pp. 1429–1439, 2010.
12. H. Thimbleby and A. Gimblett, “Dependable keyed data entry for interactive systems”, Proceedings of the Fourth International Workshop on Formal Methods for Interactive Systems, Volume 45, 2011
13. I. Sommerville, Software Engineering, 7th Edition.
14. A. Abran, A. Khelifi, W. Suryn and A. Seffah, “Consolidating the ISO Usability Models”, Software Quality Journal, Volume 11, pp. 325–338, 2003.
15. J. Neilson, “Usability Engineering”, 1st Edition.
16. M. Welie, G. C. Veer and A. Eliëns, “Breaking down usability”, Proceedings of Interact, 1999.
17. D. Travis, “Discount Usability: Time to Push Back the Pendulum?”Internet:http://www.userfocus.co.uk/articles/discount.html, May 2, 2003.
18. A. Abran, A. Khelifi, W. Suryn and A. Seffah: “Consolidating the ISO Usability Models”, Proceedings of International Software Quality Management Conference Springer, Glasgow, Scotland, UK, 2003