IMPROVING SOFTWARE TEST STRATEGY WITH A METHOD TO SPECIFY TEST CASES (MSTC)

Edumilis Méndez, María Pérez and Luis E. Mendoza
Processes and Systems Department – LISI
Universidad Simón Bolívar
Caracas – Venezuela

Keywords: Software testing, Test cases, Method, Test strategy, Use cases, Software test.

Abstract: An interesting difference between tests and other disciplines of the software development process is that they constitute a task that essentially identifies and evidences the weaknesses of the software product. Four relevant elements are considered when defining tests namely, reliability, cost, time and quality. Time and cost shall increase to the extent reliable tests and quality software are desired, but what does it take to make actors understand that tests should be seen as a security network? If quality is not there before starting the tests, it will not be there upon their completion. Accordingly, how can we lay out a trace between tests and functional and non-functional requirements of the software system? This Article is aimed at proposing a method that allows for specifying test cases based on use cases, by incorporating elements to verify and validate traceability among requirements management, analysis & design, and tests. This initiative originated as a response to the request of a software developing company of the Venezuelan public sector.

1 INTRODUCTION

The objective of the test discipline is to assess product quality throughout its life cycle based on a set of best practices (Leffingwell and Widrig, 2006) that include the following: (a) verification of the software product's proper operation, and (b) verification of requirements' proper implementation. Grimán et al. (2003) indicate that this discipline is not usually implemented in an organized and systematic manner. Additionally, according to Kruchten (2000), Pfleeger (1998), Pressman, (2002), and Sommerville (2000), the test execution process must be considered throughout the project life cycle to ensure a high quality product. Success of this process depends on the adoption of an adequate testing strategy. A software testing strategy comprises a group of activities that describes the steps to be taken in a test process, considering the amount of efforts and resources required for achieving proper software construction (Pressman, 2002).

But, from the perspective of a company, which strategies can be used? Which methodology or method should be adopted to determine the traceability between tests and requirements? Which methodology or method ensures enhanced verification and validation activities? Which strategy guarantees the delivery of a quality software product?

In this regard, we proposes a method that allows specifying test cases (TCs) based on use cases (UCs), as a starting point for the standardization and traceability of the software development process, thus obtaining highly profitable quality products.

2 RELATED WORK

The IEEE Standard for Software Test Documentation (IEEE829-98) provides a good description of test documents and their relationship with one another and with the testing process. Test documents may include, among others, TC Specification. (SWEBOK, 2004).

Three key aspects (Utting and Legeard, 2007) are considered for functional testing: design of the test case, application of the test and analysis of results, and verification of how the test fulfills the requirements.

Perry (2006) introduces a complete guide for effective testing process, including TCs, and it proposes a TC template containing certain aspects considered for our method, such as the use of IDs for TCs and UCs. Likewise, Lewis (2000) proposes a template for TCs that includes conditions, environment, version and system.

In recent years, a Model-based Testing (TDD) approach was proposed, which provided for the automation of the design of black-box tests. SWEBOK (2004) defines black-box test where test cases rely solely on the input/output behavior.

Some authors like Pinkster et al. (2006), state that subsequent improvement in testing quality is more than likely provided that requirements are considered as the testing basis; this is known as requirement-based testing.

Likewise, UML Testing Profile (2005) introduces the integration of concepts, such as test control, test group, and TC into the TC concept, which can be decomposed into several lower-level test cases and permits the easy reuse of TC definitions in various hierarchies.

Some benefits of the requirements/test matrix (Lewis, 2000) include: correlation of tests and scripts with requirements, facilitation of review status, and provision of a traceability mechanism throughout the development cycle that includes test design and execution.

In contrast to prior initiatives, our method includes all the aforementioned ideas for the purpose of obtaining a method that supports elements comprised in a test strategy.

3 TEST CASE

A TC is a specification -usually formal- of a set of test inputs, execution conditions and expected outputs identified for the purpose of assessing the particular aspects of a testing element (Leffingwell and Widrig, 2006): (a) TCs reflect traceability with UCs (functionality), (b) TCs include the complementary specifications of the requirements, and (c) TCs provide the system's design specifications.

All these elements ensure the compatibility of test procedures with user/consumer requirements. In practice, it is assumed that a UC itself is a TC and that the project team works on the UCs without planning the TCs in advance. As they test UCs, they intuitively assume the test data and procedures without making the need of documentation, which is rather a mistake, since TCs expand or enhance the information included in UCs. For instance, for UCs, the values or conditions of tests are not specified.

TCs are essential for all testing activities (Leffingwell and Widrig, 2006) due to the following:

- They constitute the basis for the design and execution of test procedures.
- Tests' depth is proportional to the number of TCs.
- Design and development, as well as required resources, are governed by the required TCs.

If TCs are incorrect, the system quality will not be reliable.

The method proposed herein states that testing procedures are comprised of steps, conditions, values, and expected/obtained results. Moreover, the testing procedure may be automated through test scripts. All the aforementioned concepts allow for visualizing the test scope: What will be tested? How, who, when, and what for? Once all TCs are executed, the results should be fully disclosed for the purpose of determining whether the acceptance criteria defined by the user were satisfied upon system validation.

In the following section you will find more details of the proposed method.

4 METHOD TO SPECIFY TEST CASES (MSTC)

The proposed method consists in creating a set of TCs from a UC, since it is assumed that software behavior must be tested based on requests or requirements. Moving from a UC to a corresponding set of TCs implies a reasonably wide and nontrivial process. Leffingwell and Widrig (2006) describe four (4) steps for achieving this process. Such steps indicate what it is to be done, but they do not explain in detail how to do it. Certain aspects that were not expressed in writing were gathered and proposed through the MSTC, based on bibliographic review and our experience. Considering those steps proposed by Leffingwell and Widrig (2006), we intend to provide a method for specifying a set of TCs from a UC.

The MSTC contribution consists in the incorporation of tests' traceability elements as to the entire development process and enhancement of the testing strategy while regulating this process. The development process is then structured in phases, activities, roles or individuals involved and artifacts generated. Likewise, it helps documenting ideas that were issued prior to tests, and explaining how TCs were generated. This is useful for verify *tests traceability with respect to previous phases, and requirement, analysis and design disciplines, and it guarantees the organization's knowledge management regarding quality assurance.*

This method includes the 4 roles proposed within the test discipline: test manager, test designer, test analyst and tester. Each phase is described in the following paragraphs, but due to space limitations, the researchers' contribution will be highlighted in italics.

It should be mentioned that MSTC is activated by the test analyst once UCs narratives are verified and upon system functionalities' approval by the stakeholders.

4.1 Phase 1: Scenarios' Identification

Activities in this phase to be conducted by the test analyst are as follows:

1. Scenarios are identified based on the UC narratives and considering specific scenarios for each UC. The regular flow, each alternate flow or a combination of both represents a scenario susceptible of being executed and tested. *Consequently, the first scenario will always evoke the regular flow of that particular UC. The relations between the* UCs and the scenarios may be one-to-many.
2. *Graphical representation of the sequence of events for each UC: A*s shown in Figure 1, this allows for abstracting events taking place in a UC, i.e. the regular or basic flow and alternate flows, and it helps *visualizing the potential combinations that would represent a scenario, since it determines the point at which the basic flow occurs and also what happens upon alternate flow activation. Then, the UC is completed or returned to the basic flow.*
3. *Verification that all alternate flows, including their completion or return actions, were graphically represented.*
4. Illustration (as seen in Table 1) of all scenarios associated to a UC in Figure 1.
5. *Identification of all UC's scenarios, indicating regular and/or alternate flow(s) comprised within the UC.* Table 2 represents the first of 3 devices generated in the MSTC: Table: Scenarios per UC.

In this table, we may observe that the ID scenario is entered for the purpose of establishing the tests' traceability element, thus facilitating the verification and approval of the tests and related UCs. As can be seen in Table 2, IDs may include the number of UCs and scenarios.

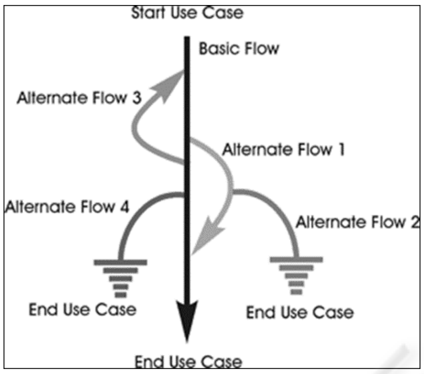

Figure 1: Flow visualization in a UC (Leffingwell and Widrig, 2006).

Table 1: Scenarios per UC (Leffingwell and Widrig, 2006).

Number of scenario	Originary flow	Alternate flow	Alternate next	Alternate next
Scenario 1	Basic flow			
Scenario 2	Basic flow	Alternate flow 1		
Scenario 3	Basic flow	Alternate flow 1	Alternate flow 2	
Scenario 4	Basic flow	Alternate flow 3		
Scenario 5	Basic flow	Alternate flow 3	Alternate flow 1	
Scenario 6	Basic flow	Alternate flow 3	Alternate flow 1	Alternate flow 2
Scenario 7	Basic flow	Alternate flow 4		
Scenario 8	Basic flow	Alternate flow 3	Alternate flow 4	

6. Verification of identification and description of all potential scenarios for each UC.

In short, each scenario represents a number of possibilities to execute a UC and prevents from testing only some potential combinations.

4.2 Phase 2: TCs' Identification

This phase takes the following activities, which should be assigned to the test designer:

1. *Ideas for new tests are organized* based on items to be tested: *functionality (UC), quality attributes, validation of inflow and outflow, databases, interfaces, etc. This will depend on the type of application, technological restrictions, scope of the project, purpose and motivation of tests, and expertise of the test team (especially the tester's).*
2. Table 3 must be filled in. This represents the second device generated from the proposed method and its data is associated to consider the TC for Scenario 02-02 (login error). Based on information generated by the test ideas, there is one TC for validating a "login error" upon entering invalid characters; one TC for logins in lowercase letters; one TC for logins lengths over 10 characters; and one TC for blank logins. *The, information related to*

all identified TCs is completed: test case ID, TC name, expected results (values, error messages, etc.), test level and test type. With respect to the TC's ID, we suggest including in the standard nomenclature determined by the organization a UC-scenario-TC structure, for instance, in order to identify that 02-02-01 refers to TC 01 of scenario 02 of UC 02.
3. It is verified that all TCs have been identified for each scenario. Then, the following phase is addressed.

4.3 Phase 3: TCs' Specifications

One of the most significant contributions of this research is the third device *(Design)* used to describe TCs in detail, as shown in Table 4: TCs' specifications (TCS). This should also be completed by the test designer and the following activities should be performed for each TC:
1. *Identification of the system's name, UC ID, requirement ID, scenario ID, TC ID and TC version.* This allows for laying out a bidirectional trace between these elements: for instance, it may determine whether all TCs were specified for the UCs and whether all UCs were already tested (tests scope).
2. *Identification of the level and type of test associated to the TC,* the information of which is generated by TC as per the scenarios table.
3. *Identification of the test environment.* The name of the company might be indicated, provided that it is the development or production environment. If the company has different environments, the environment where this particular TC will be executed should be indicated.
4. *Identification of the TC creator and tester. We recommend that two different individuals perform these activities so the testing process can be true and reliable.*
5. Indication of the TC's origin date and execution date.
6. *Identification of the conditions that should be present to execute the TC.* Which are the conditions for causing or making a user to execute a specific event or series of events? In Table 4, we observe that all functionalities associated to user's validation should be implemented. Likewise, it should be verified that data to be used for this TC has been validated and approved by the corresponding level, etc.

Table 2: Scenarios for UC002.

Scenario ID	Originary flow	Alternate flow	Alternate next	Alternate flow
02-01	Basic flow			
02-02	Basic flow	Alternate flow 1: Login Error		
02-03	Basic flow	Alternate flow 1: Login Error	Alternate flow 3: Cancel Press	
02-04	Basic flow	Alternate flow 2: Password Error		
02-05	Basic flow	Alternate flow 2: Password Error	Alternate flow 3: Cancel Press	
02-06	Basic flow	Alternate flow 1: Login Error	Alternate flow 2: Password Error	
02-07	Basic flow	Alternate flow 1: Login Error	Alternate flow 2: Password Error	Alternate flow 3: Cancel Press
02-08	Basic flow	Alternate flow 3: Cancel Press		

Table 3: TCs per scenario 02-02.

Test Case ID	TC name	Expected results	Test level	Test type
02-02-01	Login with invalid characters	*Message 20 Error: Invalid Login*	System/ acceptance	Function/ Access Control
02-02-02	Login with Lowercase letters	*Message 20 Error: Invalid Login*	System/ acceptance	Function/ Access Control
02-02-03	Login length over 10 characters	*Message 20 Error: Invalid Login*	System/ acceptance	Function/ Access Control
02-02-04	Login in Target	*Message 20 Error: Invalid Login*	System/ acceptance	Function/ Access Control

Table 4: TCs' Specification (TCS) 02-02-02.

System/Project ID/Name: SIS-PROJ	**Test Level:** System/Acceptance	
Use Case ID: CU-02 User's Validation	**Test(s) Type(s):** Function/ Access Control	
Requirement ID: *(solo para Caso de Uso No Funcional)*	**Test environment:** *AMBIENTE1*	
Scenario ID/Name: 02-02 Login Error	**Test Case´s Author:** LISI	
Test Case ID/Name: 02-02-02 Login with Lowercase letters	**Tester´s name:** *Probador 1*	
Test Case Version: v.1.	**Origin Date:** 10-01-07	**Execution Date:** 15-03-07
Condition(s) for that Test Case is executed		
The user wishes enter the system. All functions related to user´s validation have been implemented. Data to be used for the tests have been validated and approved. Certain users have been registered as valid users.		
For the execution of Test Case:		

Step	*Condition*	*Value(s)*	*Expected Results*	*Obtained Results*
It enters the login and presses OK.	Log-in attempt	aDM22	Message 20 Error: Invalid Login	√
It enters the login and presses OK.	Log-in attempt	administrator	Message 20 Error: Invalid Login	√
It enters the login and presses OK.	Log-in attempt	AdminisTRATOR	Message 20 Error: Invalid Login	√

Test Case Approval Criterion : If the expected results are achieved in a 100%
Test Case´s Decision of approval: Approved: _X___ Failed: _____ **(mark with an X the results obtained)**
Test Case´s Date of Approval: 15-03-07

7. Description of the TC procedure. This procedure comprises steps to be taken for testing the UC scenario through the TC approach, i.e. *particular conditions that might apply for a given step,* values used, results expected and results obtained. It should be mentioned that the latter is included in the TCS table upon TC execution.

If data is not properly entered, it would not be possible to execute tests and determine the results. Supplementary specifications should be followed to determine the performance measures (minimum and maximum), inception valid ranks, interface protocols, among others.

8. *Indication of TC's approval criterion.* As observed in Table 4, the main criterion is that all expected results must be 100% achieved.

9. *The test analyst and designer verify that all TCs have been properly specified.*

4.4 Phase 4: TCs' Execution and Approval

Activities in this phase include the following:

1. *Verification that the environment and conditions to execute the TC are appropriate.* Individuals involved in these tests must cooperate to this procedure.
2. *The test manager and designer must authorize the test cycle activation.*
3. *The tester executes all TCs and enters data of the results obtained into each TCS Table.*
4. *The test manager decides on the approval/rejection of a TC based on the criterion established, and it should also indicate the date of approval and, in certain cases, its signoff.*
5. *The test team verifies whether the test cycle completion criterion was met* to decide on the test cycle's fully approval or request the application of additional tests on certain TCs for a subsequent test cycle, until all acceptance criteria are satisfied.
6. *The test team keeps all deliverables and posts the results of the test cycles,* changes, etc., obtained during the testing process.

4.5 Phase 5: Recording and Analysing Results

The objective is to maintain and improve the TCs asset. This is important especially if the intention is future reuse, in subsequent test cycles or to other software products. This phase is centered on: (a) determine the minimal set of additional TCs to validate the stability of subsequent Builds, (b) remove TCs that no longer serve a useful purpose or have become economic infeasible to execute, (c) conduct general maintenance of and make improvements to the maintainability of TCs automation, (d) explore opportunities for reuse and productivity improvements, (e) maintain test environment configurations and test data sets, and (f) document lessons learned –both good and bad practices–discovered during the TCs execution.

4.6 Strategy Checkpoints

To ensure the proper method application and strategy accomplishment, the following checkpoints are required:

- Phase 1: (a) Is there any matrix of scenarios per each system UC?, (b) Check that all scenarios in the corresponding UC have been included in the matrix of scenarios per UC, (c) Check for completeness of IDs of UCs and TCs and their correspondence with the nomenclature proposed.
- Phase 2: (a) Is there any matrix of TCs per scenario?, (b) Check for completeness and accuracy of IDs, names, expected results, tests levels and test type for each TC matrix per scenario
- Phase 3: (a) Is there any TC table with specifications for each TC identified in the prior stage?, (b) Check traceability among IDs of the TCs, UCs, their requirements and scenarios, (c) Check for accuracy and completeness of all items indicated in the TC specification table, (d) Were approval criteria for each TC specification validated?
- Phase 4: (a) Were results from the execution of TCs through field fill-in, namely obtained results, approved/failed, date of approval, date execution, documented?, (b) Was the test cycle completion criterion indicated in the Test Plan document?, (c) Were the results from tests and changes applied during the testing process delivered and posted?

5 RESULTS DISCUSSION AND FUTURE WORK

The MSTC method proposed was used for 4 projects, thus obtaining significant results as to: (1) quality of the developed products. Upon completion of the construction phase, software systems had already reached 90% of the expected quality; (2) the largest project implemented 51 UCs and required the documentation, execution and approval of 460 TCs. The TCS Table can be used to define TC procedures associated to non-functional requirements; and, (3) this experience established a precedent for future projects, and defined management indicators that may reflect, for instance, the average number of TCs per application. Currently, MSTC is being used by other public and private sector organizations working in 16 projects; therefore, the following step in this research should be posting the results from the method application at each organization.

6 CONCLUSIONS

This Article described a method for specifying TCs based on UCs, by incorporating elements to verify and validate traceability among requirements management, analysis & design, and tests. In addition, it evidenced that test costs might be reduced at a mid- and long-term, since we may resort to non-specialized testers, provided that what will be tested, when and how? is clearly defined in advance.

REFERENCES

Kruchten P., *The Rational Unified Process as Introduction* (2nd Edition, Addison – Wesley, 2000.

Leffingwell D. and Widrig D., 2006. *Managing Software Requirements, a Use Case Approach* (Second Edition, Addison-Wesley, Pearson Education, 2006.

Grimán A., Pérez, M. and Mendoza L., 2003. Estrategia de pruebas para software OO que garantiza requerimientos no funcionales. *III Workshop de Ingeniería de Software, Chillán*, Chile. 1-10.

Lewis W. 2000. Software testing and continuous quality improvement 000 by CRC Press LLC Auerbach is an Publisher of CRC Press LLC.

Perry W. 2006. Effective Methods for Software Testing. Wiley. Third Edition.

Pfleeger S. 1998, *Software Engineering, Theory and Practice,* Pretice-Hall.

Pinkster I., Burgt B., Janssen D. and Veenendaal E. 2006. Successful Test Management. Springer and Logicacmg.

Pressman R. 2002. *Ingeniería del Software: Un enfoque Práctico,* McGraw Hill, 2002).

Sommerville I. 2000. *Software Engineering*. Pearson Education.

SWEBOK. 2004. Guide to the Software Engineering Body of Knowledge 2004 Version. IEEE Computer Society.

UML Testing Profile Version 1.0 formal/05-07-07. This is a testing profile for UML 2.0.

Utting M. and Legeard B. 2007. Practical Model-based Testing. Morgan Kaufmann and Elsevier Publisher.