Transactions on Information and Communications Technologies vol 9, © 1994 WIT Press, www.witpress.com, ISSN 1743-3517

Testing transactional applications: a practical experience

G. Marzano & P. Romanazzi
Software Quality Laboratory, Tecnopolis CSATA NOVUS ORTUS, 70010 Valenzano (BA), Italy

ABSTRACT

This paper reports a practical experience of software testing accomplished by the Software Quality Laboratory of Tecnopolis Csata Novus Ortus in co-operation with Datamat S.p.A. The objective of the joint project was twofold: the definition and development of a formal test suite, the functional testing of a transaction processing software in the stock exchange application area .

With respect to the first goal we had the need of define a method for building a complete test suite in terms of: specification of test environment, test process, test documentation, criteria for test derivation. The main result of such experimentation was to allow the replication of the whole test suite in a different target site.

The VALID toolkit was chosen as the kernel for the test environment. This tool allows the validation of systems utilising a GUI based on the X window environment, against its specifications using a black box testing method (Jason, Ritter [4]). For the test process definition and test documentation we have taken as reference the IEEE standard 1012 (Software verification and VALIDation plans), IEEE standard 829 (Software test documentation).

In relation to the second goal we have built a library of test procedures formally described with the VALID test language and reaching the full coverage of the functionalities of the application under test. The experience in applying the method developed during this project together with the technology used, has shown that this approach is valuable in reducing testing time and costs especially during regression testing in the software maintenance phase.

INTRODUCTION

Testing is everywhere acknowledged as an important part of the development process. Beyond this point, in the software engineering community there is a

live debate on how should testing be done, on the techniques or methods to apply, on exploitation of testing tools. In this paper we'll not try to answer these questions, but just to explain how functional testing was done in our experience on a transactional application in the stock exchange area (Sherer [3]).
We hope these results can help in some way others who are facing with software quality and cost problems.

THE ENVIRONMENT

The testing activity has been experimented on a SUN-4 machine hosting both the implementation under test (IUT) and the test tool. The IUT was a transactional application in the stock exchange application area. ORACLE was the database management system used by the IUT.
VALID is a toolkit for the Verification and VALIDation of Software and Embedded Systems.

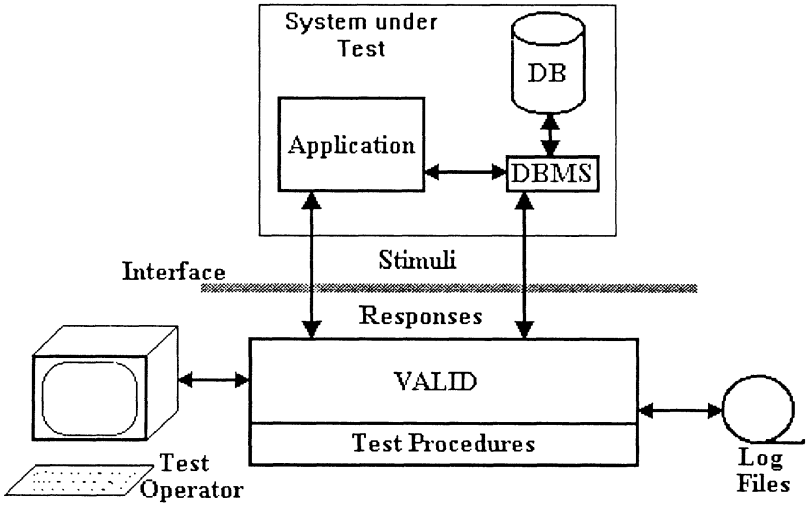

Figure 1: The environment.

By means of a flexible communication channel VALID exchanges data with the IUT, and so it constitutes a good support for black-box testing (see fig. 1). Actually it is possible to send stimuli to the application and record the system reactions. Since VALID toolkit utilizes an X-window for capturing and sending information from and to the IUT, a standard X-window terminal was opened through which the IUT was run. The ORACLE interface feature of the VALID toolkit was used for consistency testing of the database.
Some of the main features of the VALID toolkit are: availability of a test description formal language used to build test procedures, its capability of automate test procedure execution and generate test reports, cross referencing towards the relevant items in the project (such as functional requirements), recording and playback of all the interactions with the system under test.

TEST DERIVATION

Traditional testing theory has focused on selecting tests that have the highest probability of detecting the most errors (Sherer [2]). Current testing procedures attempt to cover, or exercise, as much of the software functionalities as possible to detect as many errors as possible. Expecially for black-box testing the problem of deriving complete test suites is still object of many theoretic studies (Musa, Ackerman [5]).

One of the major drawbacks of black-box testing is its dependence on the specification's correctness, which is usually not the case at the present. In fact the test designer needs a correct and complete software specification in order to generate complete test suite.
Very often, the system to be tested is delivered to the test laboratory with a very limited documentation or, at the best, a detailed user manual. Since it is unrealistic to ask the client to provide a well defined software specification document for testing purposes, the test designer has a very limited basis from which derive tests.
In this experiment we tried to integrate the user-manual specifications, as it was the only formal document the client provided, with the knowledge hidden in the system designers and developers minds.
In conjunction with traditional black-box tests we defined tests for database consistency checks using the VALID standard interface to ORACLE DBMS.
This approach, which is not much close to the academic topics on how to derive a complete test suite, showed itself to be very successful and the derived test suite was very appreciated by the client.

Starting from the process described above, the Laboratory derived the tests in order to fulfil the following criteria, which where also used to group tests into homogeneous groups [9].

Setup tests
Tests at a low level of complexity aimed at verifying the correct IUT (Implementation under test) setup and at proving that the connection with the test machine and the VALID tool is active and running.
These tests may have a kind of similarity in HW component testing with the calibration of the test tools.
Usually a little share of tests falls within this test group.

Basic Behaviour tests
Tests at a high level of complexity that reproduce a brief work session till the activation of the specific function to be tested. Even if, for its own definition, these tests are able to stimulate many IUT functionalities, each test is aimed at testing only one specific IUT behaviour.
Stress tests, if required by the user, also fall within this test group. Recycling peaces of tests already defined, it is usually possible to define an effective stress

test (i.e. the insertion of one numeric entry in the database and subsequent grandtotal).
A major percentage of test within the test suite fall within this group.

Parameter Variation tests
These tests are aimed at stimulating the software behaviour under the variation of functions parameters.
System functions are invoked repeatedly each time with different parameter values.
For numeric parameters the minimum and maximum allowed values are tried and usually one or more values within the range depending on the criticality of the parameter (e.g. test of parameters which may vary the complexity of the computation needs a wider coverage)
For all enumerative parameters the widest coverage is required in particular when a specific value may change the behaviour of the system (i.e. calculate totals "T" or grandtotals "G").

Invalid Behaviour tests
These tests check the behaviour of the system under an invalid or unexpected request. An invalid behaviour could be triggered by wrong parameter's value within a legal function. It is also possible to invoke an incorrect sequence of functions or try a forbidden operation (e.g. Modify an item where updating is disabled). The expected behaviour from the system is, for all these test, a refusal of the triggered function preferably with an informative message to the user. It is also possible to test error recovery mechanisms, if they are implemented in the IUT.

TESTING APPROACH SPECIFICATION

As concerning the testing process specification, during the project there were two main phases: a first one for test process specification and experimentation, and a second one for test approach exploitation.
At the beginning of the project, we had to face to questions such as: what kind of approach apply for the definition of the test procedure and test sequence starting from the functional knowledge of the application to be tested, what kind of documentation we should produce, what kind of formalism we should use, how to apply productively the testing technology to our applicative context.
In this initial phase we had some goals such as: to experiment different kinds of specifications of the test sequence/procedure, to explore the different implementation approaches of test sequence/procedure.
To reach these objectives we decided to implement the first test sequences and procedures following a spiral test process model.
Each turn of the spiral process include the following activities: starting from the functional knowledge of the application, derive and implement the test sequence and procedures, specify test sequence/procedures, analyse and solve all the problems founded, estimate the test development solution used, improve our testing process definition. We mean for test sequence a succession of

functionalities commonly activated by the end user formally described with the VALID language, for test procedures a formal description of the necessary commands and user actions along to a description of the expected visible results of these actions for testing a user functionality, and for test case the data to be passed to a test procedure.
The experience gained in this first phase together with the references of IEEE standards 1008 and 1012 has allowed us to define a more mature testing approach exploited in the second phase.
This approach is composed of 4 main activities: test planning, test specification, test implementation, description of test result.
The test planning activity aims at building the test plan document which specifies the scope, approach, resources, and schedule of the remaining test activities.
In particular the structure of the test plan document follows that one proposed by the standard of IEEE n. 829 and is composed of the following information items:

-Identifier
It specifies the unique identifier assigned to the test plan document.

-Introduction
It summarises the software items and software features to be tested.

-Software to be tested
It identifies the software to be tested and its version /revision level.

-Features to be tested
It specifies all the functionalities of software to be tested and for each functionality identifies the set of test sequences dedicated.

-Approach
This item describes the whole approach exploited for testing the entire application but also some particular approaches used to test some functionalities or combination of them and explain the reasons.(In our experience we generally used a "black box" testing technique but for some critical functionalities we used a "white box" technique).

-Pass/Fail Criteria
It specifies the criteria to be used to determine if the software functionality has passed or failed testing.

-Suspension/resumption Criteria
It specifies the criteria used to suspend the test and describe the sequence of activities to perform for resuming the test session .

-Setup activities
It identifies all the steps needed to start up the testing sessions.

-Test environment features
It specify all the features of the testing environment such as what is the host machine, what version of O.S., what version of X windows, etc.

-Temporal schedule.
This item describes accurately the time foreseen for each testing tasks and the associated responsibilities.

The test specification activity aims at describe in an informal but complete way the test sequences/procedures. This phase is very important since it produces the test functional specifications which will allow to derive the test sequences/procedures to be implemented and executed in the next phase.
The test specification precisely describe the set of test data to apply i.e. the test cases , the logic sequence of operations, the expected results.
The output of this phase consists of two documents: a test cases specification document, and a test sequences specification document. In the first document each test case is described in terms of:

-Identifier
A unique identifier associated to each test case.

-Functionality
The name of the functional requirement tested by the test case.

-Precondition
It is a state of the system under test before the execution of the test case such as the correct execution of previous test cases.

-Input data
It specifies the set of stimuli to be sent to the system under test by means of the test procedure.

-Output data
It describes the expected responses of the system after the previous stimuli.

As concerning the test sequence specification, it aims at specify the steps necessary in terms of sequence of test cases/procedures for testing a particular feature of the system. Each test sequence is defined with the following information items:

-Identifier
A unique identifier associated to each test sequence

-Precondition
.It is a state of the system under test before the execution of the test sequence.

-Objective
It states the testing goal of the sequence

-Test steps
It specifies the flow of user functionalities or system screens called by the test sequence. Each functionality there is associated to the test case/procedure identifier that tests it.

After the functional specification of the test, the project team began to generating tests for each system's feature to be tested. At first tests were run manually and the components responses were checked with the behaviour described in the test specification. This initial activity was recorded by means of the VALID X-recording tool. During this initial activity, we asked the development team to reproduce on the system under test several real work session highlighting, wherever possible, critical points within the software functions.
The tool captured the user/system interaction and translated them into a sequence of VALID language statements composing our initial test procedures.
These initial test procedures were later parameterized to make them easier to be used with different sets of input/output data and for later maintainance.
These set of test procedures forms the test suite. Since the application under test was not stable but undergone subsequent modifications, the test suite demonstrated its usefulness during the regression testing activity. Regression testing is important since changes and error corrections tend to be much more error prone than the development of the original code.

The VALID toolkit was of primary importance in this phase. The VALID test execution tool gave us the opportunity to play back individual tests, a subset of the test suite, or the entire test suite. The regression tests could be performed automatically without operator intervention to ensure that the changes did not introduce new errors.
Any inconsistency between the resulting behaviour and the behaviour described in the specification were reported by VALID in the test log for further investigations. Only those tests that failed would need to be rerun under observation, saving a lot of man-hours. Moreover the formalization of the approach, would guarantee the same quality of testing even in the future.
At the end of this testing activity the test team developed a test summary made up by all the test logs and by two test summary matrix as shown in table 1 and table 2. In the first matrix the columns represent the test sequence identifiers, on the rows the test cases identifiers are listed, and within the cells there are the number of call and the number of failure separated by a comma.

	TS_01	...	TS_n	TOT
TC_fxx_01	1,0	...	4,1	8,3 + ...
TC_fxx_02	1,1	...	7,0	13,4+ ...
...	...	...	...	...
TC_fxx_n	7,3	...	5,2	13,2+...
TOT	9,4	...	16,3	

Table 1: Example of Test Sequence - Test Cases cross reference table.

In this second matrix is shown a cross referencing with the system functionalities, indicating on the columns the test sequence identifiers, on the rows the system function names or identifier, and on the cross the number of calls each test sequence makes for each system function.

	TS_01	...	TS_n	TOT
Fx1	1	...	4	8+ ...
Fx2	1	...	7	13+ ...
...	...	...	...	...
Fxn	7	...	5	13+...
TOT	9	...	16	

Table 2: Example of Test Sequences - Function mapping table.

CONCLUSIONS

Economic competition often forces developers to release applications before they have been adequately tested. This lead, as the time goes by, to a more difficult and a slower corrective, adaptive maintenance activity, and to a degradation of the overall software quality. We must use testing technology for helping in delivery a quality software product in a timely fashion for minimum cost (Dunham [1]).

VALID environment gave a great help allowing the recording of sample test sessions and the collection of test results with very limited effort.

The technique used to derive the test suite, execute the test and report the results has shown itself to be successful and user appreciated. The automatic execution of the test suite gives the opportunity to run all or part of the test in a batch way and makes life easier to the test operator to verify the expected results. The automatic execution also allows the user to ask the laboratory quick regression tests sessions with the great opportunity to test the implementation whilst in the maintenance cycle.

The maintenance of the test suite and the variations triggered by modifications of the IUT are made easier by the detailed test specification adopted in this experiment together with the high level of the test specification language used.

REFERENCES

1. Dunham, J. R. "V&V in the Next Decade" IEEE Software May 1989

2. Sherer, S. A. "A Cost-Effective Approach to Testing" IEEE Software March 1991

3. Sherer, S. A. "Measuring the risk of Software Failure: A Financial Application" Proc. Int'l Conf. Information Systems, Boston, Mass. 1989

4. Prather R.E.
"Theory of Program Testing - an Overview", The Bell System Technical Journal, 62 (10)ii, December 1983.

5. Musa, J. D. and Ackerman A. F. "Quantifying Software Validation: When to Stop Testing?" IEEE Software May 1989

6. Myers, G. J. "The Art of Software Testing" Wiley & Sons, New York 1979

7. IEEE standard 1012 "IEEE Standard for Software Verification and Validation Plans"

8. IEEE standard 829 "IEEE Standard for Software Test Documentation".

9. ISO 9646 "Information Technology- Open System Interconnection- Conformance testing methodology and framework".

10. Parrington, N and Roper, M "Understanding Software Testing" Ellis Horwood Ltd. 1989

11. Probert, R. L. and Ural, H. "High-Level Testing and Example-Directed Development of Software Specification", Jnl. of Systems and Softeare, November 1984

12. Howden W.E.
"The Theory and Practice of Functional Testing", IEEE Software September 1985

13. Howden W.E.
"A Functional Approach to Program Testing and Analysis", IEEE Trans. Software Engineerig, SE-12 (10) 1986.