9

March, 2010

testing experience

The Magazine for Professional Testers

printed in Germany

print version 8,00 €

free digital version

www.testingexperience.com

ISSN 1866-5705

Advanced Testing Techniques

iqnite GERMANY 2010 Conferences Special

ONLINE TRAINING

English & German (Foundation)

ISTQB® Certified Tester Foundation Level

ISTQB® Certified Tester
Advanced Level Test Manager

ACCREDITED ISTQB TRAINING PROVIDER

ISTQB International Software Testing Qualifications Board

Díaz Hilterscheid

te testing experience

Editorial

Dear readers,

the call for articles for this issue was a very big success. I never thought that we could achieve this kind of response. We got more than 400 articles and selected around 30 for you.

We also expect to get a great number of responses for the next issue about "Performance Testing".

Please note the deadline for the proposals.

We have seen in the last weeks once again that quality and testing are quite important for the awareness of the market.

Toyota is an example how quality problems cost a lot of money and how they can be dangerous for human life. This matter will help to show the work of QA specialists in a better light, but I don't think that it will be enough to achieve the necessary support from the management. I think that they still think quality costs money. The day to day life teaches us that poor quality costs even more money!

Google Buzz is also an interesting issue from the QA point of view. A big company, with a lot of good professionals and really taking care of QA, shipped a new product with some problems. STERN - a German newspaper – said that Google should think about changing the name form "Buzz" to "Mess". In my opinion it was a mistake to launch a product that had some kind of difficulties specially when you plan to make a strong impact on the market, but we are not fully informed about the real situation behind the curtains. I think that behind the decision to deliver the product could be the old story: When should we stop testing? Did we test enough? Is the product ready for the market?

I wonder if they did have a risk based business analysis. Did they know the expectations of the market and did they consider the reaction if they would have a "fehlstart", a launch failure?

If you know what happened, send me an article about it! We are happy to inform our readers.

We have started some cooperations like the year before with some conferences, training providers, etc to get the best for our readers.

In this issue you can see a special section on the ignite-conference in Düsseldorf, one of the biggest conferences in Europe. You, as a reader, get a discount to attend it. Go for it!

The Sela Group and ImproveQS are offering some discount for training in the Netherlands, Canada and Israel. Please use these discounts.

We are very proud to support the CINTEQ in Sao Paulo, Brazil and also the TestIstanbul – where I'm speaking – in Istanbul, Turkey. Happy to be in this countries.

In June 2010 we have the Testing & Finance conference in Frankfurt, Germany. It is the conference for the testing professionals in the finance industry. I hope to see you there. Let's go for a chat and a coffee or a beer!

Last but not least I'd like to thank all the authors for their articles and columns and of course the supporting companies for their ads. Thank you. We have the largest "testing experience" ever!

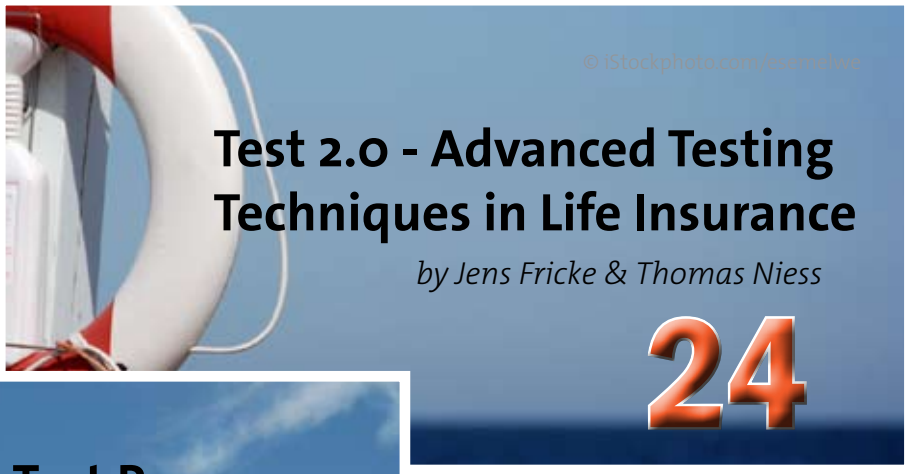
Please don't hesitate to contact me, if you have any question.

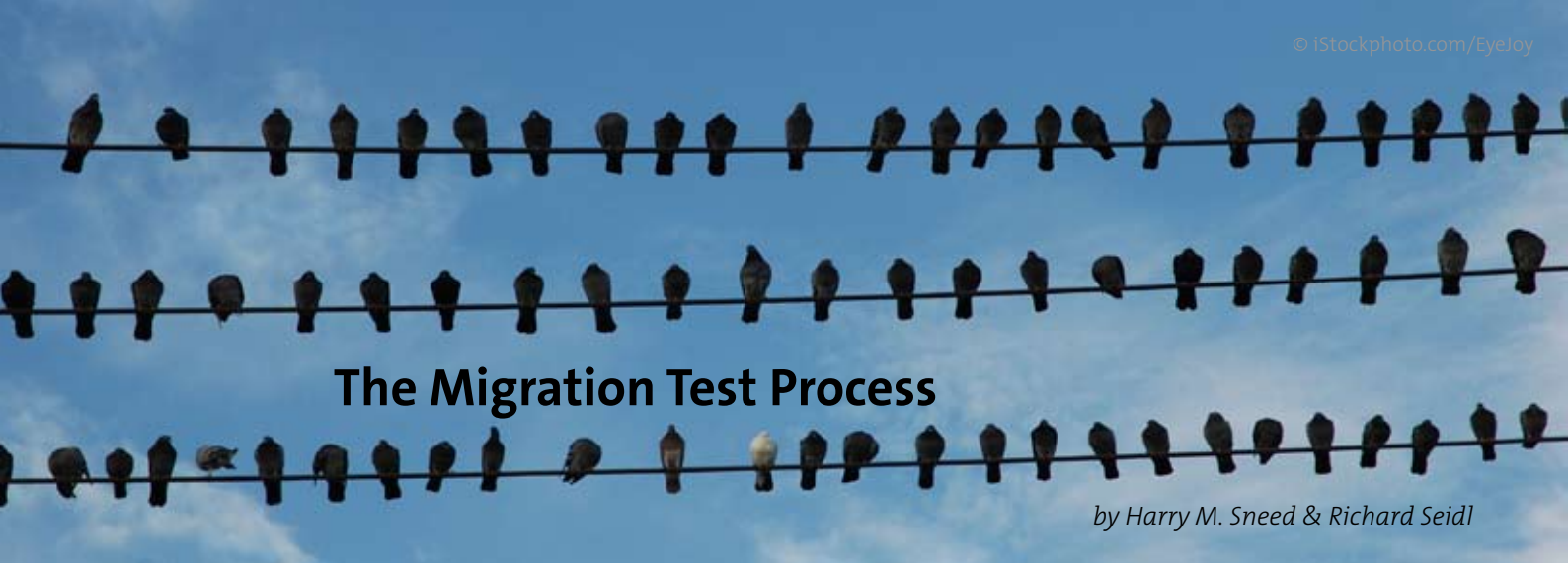
Enjoy reading!

Best regards

José Díaz

Contents





The Migration Test Process

by Harry M. Sneed & Richard Seidl

A migration test must begin by measuring the source code to be migrated. This is necessary in order to know exactly what has to be tested and what effort will be involved. Only after the software has been measured, is it possible to perform the next step, which is to plan the test.

The second step, test planning, includes the setting up of a budget and a time schedule as well as in detailing the tasks to be performed and the results to be delivered.

The third step is to design the migration test. It is important here to identify each and every test object, i.e. user interface, database table, system interface and report used by the system. The test procedure, i.e. the test steps and their sequence, also have to be defined.

The fourth step is acquiring the test data. Normally, the test data for a migration is taken from production and possibly scrambled to avoid privacy problems. Test data is acquired by executing the old system in a controlled environment and recording both the inputs and the outputs as well as the database states. The problems lie in handling such large volumes of data, the many imports, exports and reports and in recording the user interactions.

In the fifth step, the migration test is executed with the data acquired in the fourth step. Since there are no detailed test cases as there are when requirements are available, the test case is usually an online transaction or a batch process. The amount of input data that goes into such a test case can be great, so it could be difficult to specify. The same applies to the outputs of regression test cases. As a consequence, the migration test has to be automated. Test execution in a migration project is mainly about handling tools.

In the sixth step, the results of the migration tests are evaluated to determine if the migrated system is performing as expected, that is, exactly as the previous system was. This is determined by comparing the results of the new system with those of the old system. Any deviation is a sign of non-compliance.

In the seventh and final step, the test coverage is measured to determine whether or not the code has been adequately covered. This will depend on the goals set in the test plan. If so, the test can be terminated, if not the test has to be repeated until the goals have been met. The test goals, e.g. transaction, code and data coverage, are the basis of the test service agreement – TSA. (Figure 1)

Code Measurement

To quote Tom DeMarco, you cannot plan what you cannot measure [2]. Therefore, to plan the test of a migration, you must begin by measuring what is being migrated, namely the code. It is the code which is being transformed into another form. Should the data have to be migrated as well, then the data structures will

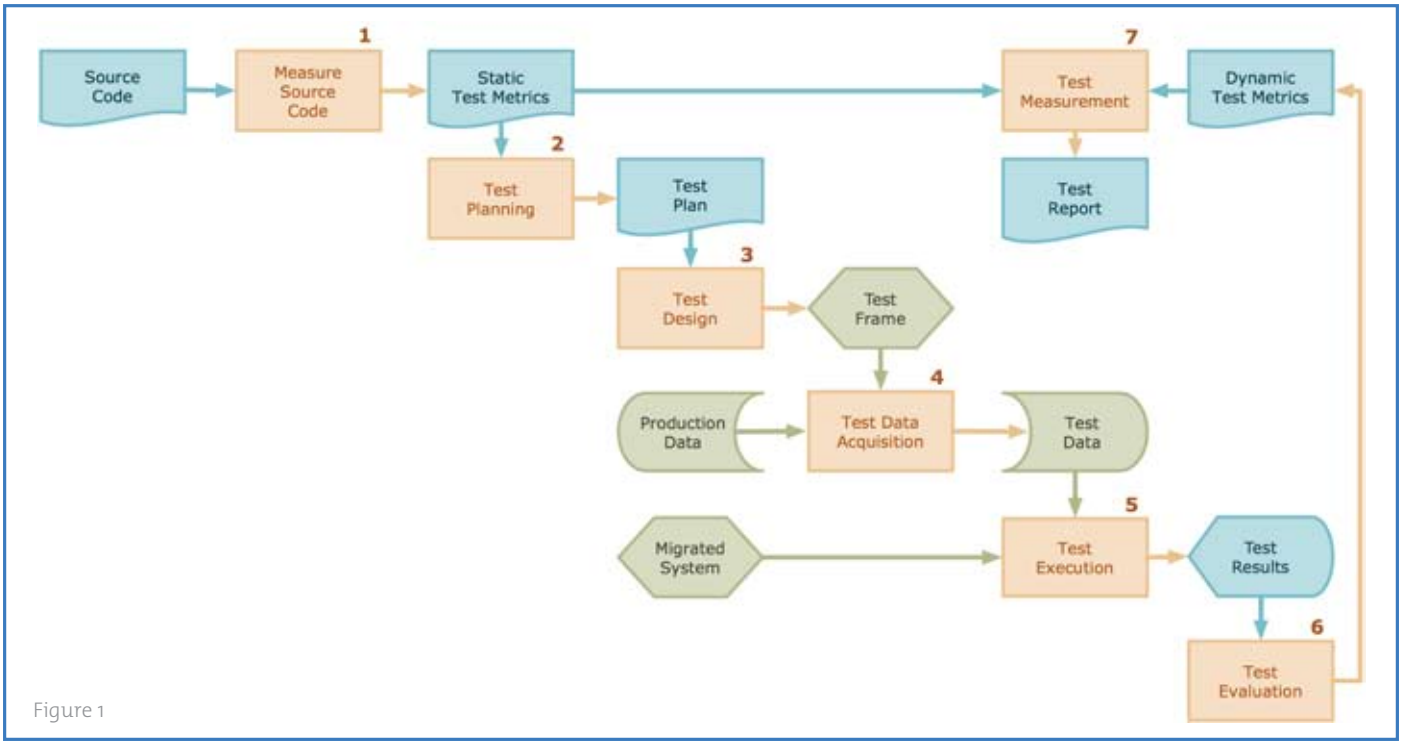


Figure 1

also have to be measured.

There are many aspects of the code which can be measured, like control flow complexity, data usage complexity, module interaction complexity, reusability, maintainability, conformity etc. What one measures depends on what information one needs to accomplish the mission. In this case, we are concerned with knowing how many code units, data base tables and data interfaces we have to test in order to demonstrate functional equivalence. These metrics are extracted with static analysis of the source code, the data base schemas, the user interface definitions and the system interfaces. The resulting measures indicate how many data interfaces with how many data tables are required to test each component. The static analyzer tool also computes the testability of the system on a rational scale of 0 to 1 [3].

Normally, the code measurement requires no longer than a week, as experienced in past projects, provided one has the appropriate measurement tools. Fortunately, the authors have tools which cover both, old and new languages. In most migration projects, the previous system is coded in an old language like PL/I, COBOL or C.

Planning a Migration Test

Planning a migration test entails not only setting the test goals and allocating the test resources, but also estimating the time and effort required to make the migration test. Test effort here is determined by the number of test cases that have to be tested plus the number of test objects that have to be validated. A test case can be an online transaction, an event or a batch process. They are derived from the production usage profile. A test object can be a user interface, a file, a database table, a system interface or a report. They are taken from the production data. Together they add up to the test volume. It is necessary to divide this volume by the test productivity measured in previous projects to come up with the raw effort required. This can be adjusted by the product type, the product testability ratio, the project influence factors and the test repetition factor [4].

The result of test planning is a test plan according to the ANSI/IEEE standard 829 plus a test effort and a test time estimation. The two results are the basis for a price offer to the user to test his migrated system. In the offer, particular emphasis is placed on the test goals and the test end criteria. They must be met in order for the customer to accept the test. For that reason they must be stated in an easily measurable form such as the ratio of the test cases executed and the test objects validated. The final version of the test plan can be a standard Word document with the IEEE-829 table of contents [5].

Designing a Migration Test

To design a migration test, one must consider what and how it will be tested. What has to be tested is described with the test cases and the test objects. The testers have to analyze the current production processes in order to obtain them. Every relevant variant of every existing transaction is a potential test case. Unfortunately, it is not possible to rely on a legacy system's documentation, since it is hardly ever up to date. In many cases there is none at all. The best way to construct the test is for the tester to sit down together with the end users and record how they are actually using the current system. This amounts to a post documentation of the operational profile and accounts for the biggest part of the regression test effort.

The test objects are easier to identify, since they can be observed. They are also recorded by the measurement activity. What is important is the number of data items – fields, columns, tags, widgets, etc. – that each data object has and their representative value ranges, as taken from the production data.

The how of migration test design is concerned with how the test objects are to be acquired and validated, and how the test cases are to be executed. The test design must foresee where the test objects are taken from and by what means the test cases can be recorded. The test design itself should be in a structured, semi-formal format. For example, Excel tables, graphic tools and XML-documents can be used. (Figure 2)

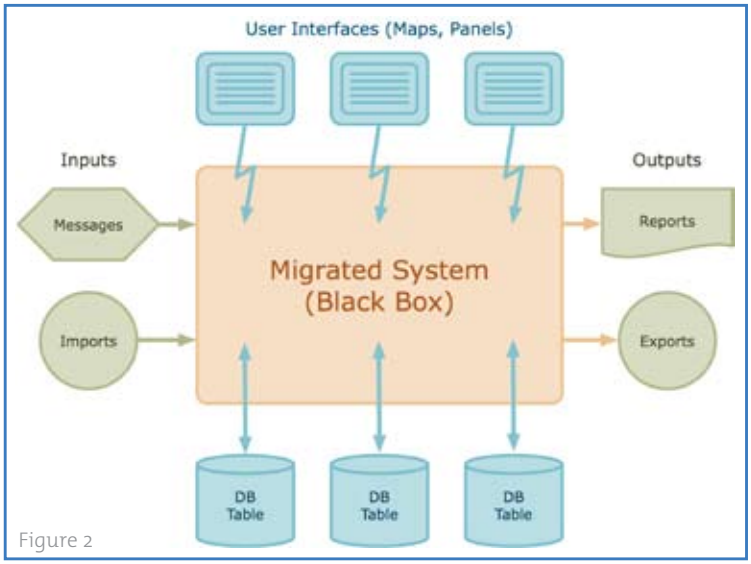


Figure 2

Acquiring Test Data for a Migration Test

As opposed to testing a development project, one does not have to generate data to test a migration project. The test data can and should be taken from production. The question here is what data to take. It is too much to copy the whole production environment. In the test design those transactions, events and batch runs are identified which are representatives of others. These must be executed in a controlled environment using the existing system. Prior to each test, the images of the user panels, the input files and the database contents are archived. After the test the new images of the same objects plus any additional objects such as output messages sent and reports printed are recorded in the same way. This will lead to a huge mass of data which has to be managed. In previous projects, often a special server was required just to archive all these test data.

It is here that test tools can be very helpful, a) in recording the user interfaces and b) in administering the many old and new images that have to be kept. It is here, where a capture/replay tool can be helpful. Where one was not available, the authors helped themselves in the past by recording the production user interface images with a simple screenshot and using this as a specification for testing the migrated system. Whether with or without tool support, the problem of data acquisition for migration testing is mainly that of administering huge volumes of recorded data. Anyone planning to test a migrated system of any size must be prepared to handle this [6].

Executing the Migration Test

What is unique about migration projects is the amount of testing relative to the other activities. When systems are developed, they are being tested module by module, component by component and system by system over a long period of time. Each module has to be designed, coded and tested, and these activities are highly intertwined, so it is not so obvious how much testing is going on. Only the system test at the end stands out, as a pure testing activity. Not so with a migration project. In a migration project whole systems have to be tested at one time, and since most, if not at all, of the code transformation is done automatically by persons totally unfamiliar with the functionality of the system, there is no way to know where errors will occur. If something is transformed wrong, they most likely will not even notice it. The whole burden of demonstrating correctness and locating transformation errors is placed on the testers. Thus, 60-80% of a migration project ef-

fort is devoted to testing. If the test is not automated, it will be impossible to complete it in any reasonable time.

Before starting the test, the migrated code should be instrumented by placing probes in every branch or at least in every method or code block. This is done to be able to determine afterwards what portion of the code has been tested. There are automated tools for doing this. Then the test can begin. The migrated system is bombarded with the data of the old transactions, one after the other. Even if a transaction fails, the next one is started. A test driver must be able to simulate the human user. After every transaction or batch process, the contents of the output panels, the output messages and the new images of the affected database tables are recorded for later evaluation. It takes too much time to validate them during the test. If there is no automat to save the output screen contents, they should at least be saved manually as a screenshot and stored for later comparison. The emphasis here is on testing as much as possible without interruption. If anomalies occur, they are recorded to be checked out later. There is not much place for creative testing in a migration test [7].

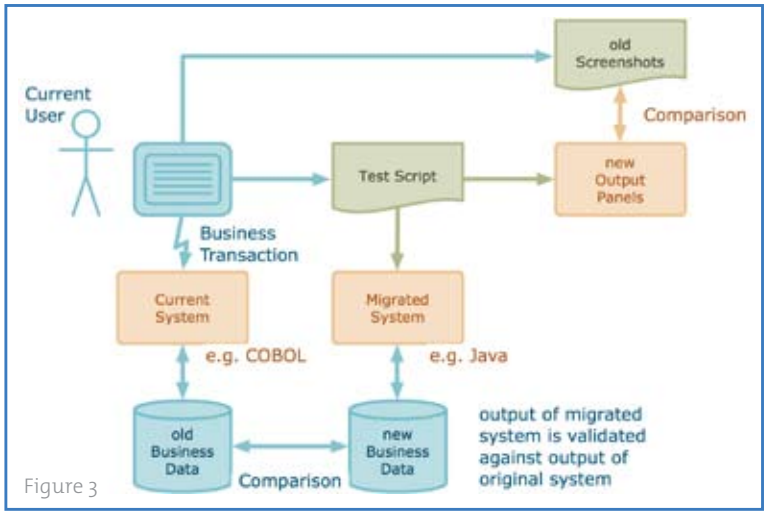


Figure 3

Evaluating the Migration Test

Only after the test execution is finished, the results of the test should be validated. For validation, the new images of the migration test are automatically compared with the images retained from the test of the old system. The new screen contents, interface data and database tables are matched one at a time with the corresponding old ones. If they are in a different format, then they have to be converted to a common one for comparison.

For this purpose, one of the authors has developed several conversion and comparison tools. The user interfaces, which may earlier have been mainframe maps or UNIX screens, are transformed into XML documents. The newer user interfaces, which may be web pages or mashups, are also converted into similar XML documents. In this way individual data elements can be selectively compared with one another, regardless of where their position on the screen is and how they are displayed.

The same is done with the databases. The contents of the old databases, whether they be IMS, IDMS, IDS, ADABAS, DB2, VSAM or sequential files, are converted to comma-separated value files. The contents of the new relational databases are also downloaded to the same CSV format. Then the two CSV files are compared with each other on a column by column basis. It is not necessary to compare all the columns, and different formats or calculated values can be checked via rules and assertions. (Figure 3)

A problem comes up in validating reports and system interfaces. The new reports or system interfaces may be in a totally different format than the old ones. Of course, the values of the individual data items should be identical. If the old billing amount was 19.99, then the new billing amount should be exactly the same. However, it could be at a quite different location on the print-out or screen.

To solve this problem, tools can be used, which extract individual data items from an arbitrary print- out, screen or text file and place them in a XML document. The names and types of the data items are extracted from the original data description in the screen language. The basis of comparison is a XSD schema. In this way data from an earlier mainframe listing can be compared with data now in XML or WSDL format.

The data comparison at the elementary item level reveals even minor differences, such as having a decimal point at the wrong position. When validating several million data outputs, such an automated comparison is the best way to recognize errors in the integrated code. The other way is to compare the execution paths through the code, but that requires tracing the tested transactions. There are also tools for this purpose, e.g. TestDoc, which not only records the paths through the code but also the times when the methods or procedures are executed [8].

Migration Test Measurement

Monitoring test execution is an important prerequisite for measuring the degree of code coverage. Since it is the code and not the functionality which is being converted, it is the code which has to be covered. Before test execution, probes or trace points were inserted automatically into the code. During the test, these probes record if and when they are traversed. This execution information is kept in a log file. After the test, the log files are processed to determine how effective the test really was.

In one report, test coverage is documented as the percentage of probes traversed. This is compared with the test coverage goal set in the test service agreement. In another report the path of each test transaction is documented. This indicates which code units are executed in what order. These execution paths may be compared with those of the original system to determine where the system went wrong. In a third report the code units impacted by a given transaction are recorded. If a fix should be made to a particular code unit – method or procedure – then this report tells you which test cases have to be repeated.

In testing a migrated system different types of errors will occur than in testing a newly developed system. In a new system functions are omitted, data types wrongly defined, business rules

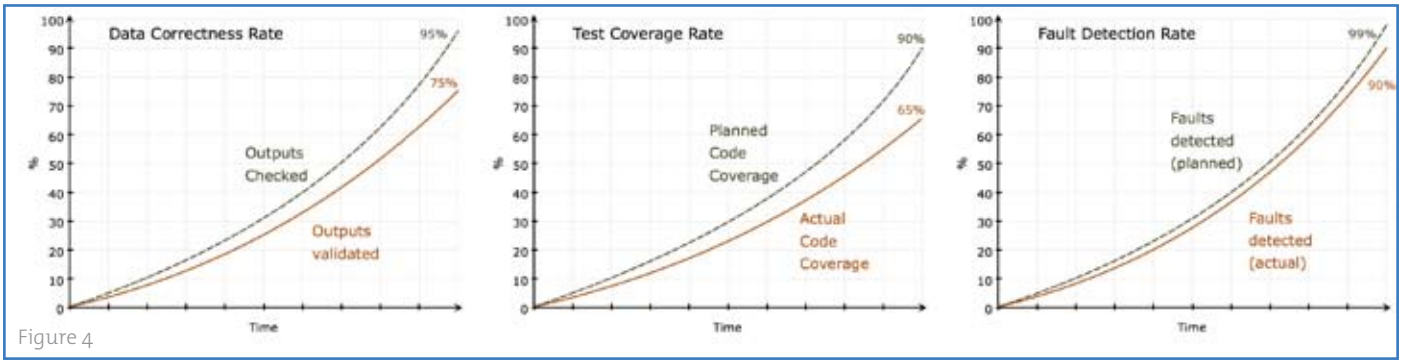


Figure 4

falsely interpreted, results calculated incorrectly, and so on. These errors are usually obvious and easy to identify. In testing a migrated system, the errors are much more subtle. Decimal places are lost, dates are distorted, data displacements are off by one and conditions are inverted. The results appear to be correct, but they are not. The only way to identify such errors is by a comparison of all data in connection with a complete coverage of all code. Since it can never be known which statement was converted wrongly, every statement has to be tested. This requires the test to be repeated many times [9].

Besides measuring test coverage, it is also necessary to measure the correctness of the output data and the percentage of faults found relative to the faults predicted. This information is used to determine when to stop testing. The planned goals are anchored in the test agreement. (Figure 4)

Summary

Testing a migrated system requires a different approach than that of testing a newly developed one. It is much more of a mass production. An enormous amount of converted code and data has to be tested blindly in a limited amount of time. To reach the goal of demonstrating the functional equivalence, nearly all of the converted code units have to be tested with a large data sample taken from the ongoing production. The huge amount of data involved and the many transactions that have to be repeated place high demands on test automation.

The experience of the authors is that migration testing demands highly skilled testers with highly sophisticated testing tools. It is not so much a question of the number of testers as it is a question of the degree of test automation. Equally important is the test organization. Such a large-scale migration test has to be meticulously planned down to the lowest detail. That requires experienced test managers, which few user organizations have. Therefore, an independent, specialized test team is necessary for a successful migration project.

Literature

[1] Onoma,A./Tsai,W.-T./ Suganuma, H.: „Regression Testing in an Industrial Environment", Comm. Of ACM, Vol. 41, Nr. 5, May 1998, S. 81

[2] DeMarco, T.: Controlling Software Projects – Management, Measurement & Estimation, Yourdon Press, New York, 1982

[3] Criag, R., Jaskiel, S.: Systematic Software Testing, Artech House Pub., Norwood, MA. , 2002

[4] Koomen, T., von der Alst, Leo, Broekman, B., Vroon, M.: TMap Next for result-driven Testing, UTN Publishers, Hertogenbosch, NL, 2007

[5] IEEE: ANSI/IEEE Standard 829 – Standard for Software Test Documentation, ANSI/IEEE Standard 829-1998, Computer Society Press, New York, 1998

[6] Fewster, M./Graham, D.: Software Test Automation, Addison-Wesley, Harlow, G.B., 1999

[7] Black, R.: Pragmatic Software Testing, Wiley Publishing, Indianapolis, 2007

[8] Sneed, H./Baumgartner, M./Seidl,R.: „Der Systemtest", Hanser Verlag, München-Wien, 2008

[9] Kann, S.H.: Metrics and Models in Software Quality Engineering, Addison-Wesley, Boston, 2001

Biography

After graduating in communications engineering, **Richard Seidl** began working as software engineer and tester in the environment of banking. Since early 2005 Richard Seidl has been test expert and test manager at ANECON GmbH, Vienna. His work focuses on planning and development of test projects mainly in the environment of banking and e-government. In 2006 he completed the Full Advanced Level Certification of ISTQB. In the following year he qualified as IREB Certified Professional for Requirements Engineering and as Quality Assurance Management Professional (QAMP). He published the book "Der Systemtest" together with Harry M. Sneed and Manfred Baumgartner, now available in the second edition.

Harry M. Sneed has a Master's Degree in Information Sciences from the University of Maryland, 1969. He has been working in testing since 1977 when he took over the position of test manager for the Siemens ITS project. At this time he developed the first European module test bed – PrüfStand – and founded, together with Dr. Ed Miller, the first commercial test laboratory in Budapest. Since then he has developed more than 20 different test tools for various environments from embedded real-time systems to integrated information systems on the main frame and internet web applications. At the beginning of his career, Sneed worked himself as a test project leader. Now, at the end of his long career, he has returned to the role of a software tester for ANECON GmbH in Vienna. Parallel to his project work, Harry Sneed has written over 200 technical articles and 18 books including 4 on testing. He also teaches software engineering at the university of Regensburg, software maintenance at the technical high school in Linz, and software measurement, reengineering and test at the universities of Koblenz and Szeged. In 2005 Sneed was appointed by the German Gesellschaft für Informatik as a GI Fellow and served as general chair for the international software maintenance conference in Budapest. In 1996 Sneed was awarded by the IEEE for his achievements in the field of software reengineering, and in 2008 he received the Stevens Award for his pioneering work in software maintenance. Sneed is a certified tester and an active member in the Austrian and the Hungarian testing boards.

Advancing testing through Simplicity

by Jonne Ratterman & Eric Jimmink

The increasing complexity of information systems makes it harder and harder for software testers to do their job well. Amazing new technological concepts have pushed the complexity even further. Developers find support in tools and integrated environments. How should testers cope with the increase in complexity? We want to show that major improvements are within reach if you learn to apply an agile testing value called simplicity to counter complexity.

Simplicity can improve the overall quality, coverage and speed of testing in complex situations. We will focus on:

- The need to collaborate
- Improving test design and test execution
- Sharing your knowledge within the team by using quick reference cards
- Breaking down the black-box

Simplicity basically means that something is easy to understand or easy to work with. That sounds very straightforward. But beware. What looks simple on the surface can actually be very complex. This is the reason why it can be very hard to successfully create something simple. Sometimes we just need a hand.

Collaboration

As a tester you might be able to picture the perfect test approach knowing that you lack the technological skills or domain knowledge needed to make it work. We can't expect a tester (or any team member!) to have extensive knowledge of all technological and business aspects. The good news is that we don't have to. It's likely that most of the knowledge and experience you need is already available within the project team. By thinking outside of the box and utilizing the combined strengths of the team members, we should be able to come up with a creative test approach and make it work.

As an example, picture a workflow management system. The functional flows through the application are long, but the risk analysis clearly shows that a calculation module in the middle of the flow requires a lot of your attention. You can tackle this problem by applying a brute-force approach, in which you work your way through the workflow time after time. You might even use a record & playback tool for support. A simpler solution is possible when working as a team. The developer can isolate the calculation module for you. Now all you need to do is enter the input parameters and check the output. You can even involve some domain experts to help set up or review the tests. This is simplicity made possible through collaboration and information sharing.

So, how do we get the other team members involved in testing? In an agile context testing is always a team responsibility. The great thing about this is that the whole team is involved in thinking about testing and quality. While testing is not a team responsibility in a traditional environment, contributing to good-quality software should be. However, it's never fair to expect your team members to offer their cooperation as an act of unselfishness. They need to benefit. If your team members help speed up testing, this means that you have some time left. You can support the developer by contributing to the (quality of the) unit tests. Invite domain experts for a test drive. One way or another, your added value to the team is in generating feedback.

Feedback

In a nutshell, testing is feedback. The principle added value of testing is the feedback that is generated by executing tests and sharing the gathered information. Feedback enables the team to analyze results and take steps to improve the quality of the product. The focus and quality of the feedback vastly improves when testing is not just performed by testing specialists. Domain experts can ensure that the right criteria are validated by testing. Developers can add focus by applying their technical knowledge of the system, placing tests at the right (low) level.

Simplicity

When people with different backgrounds work together, it is essential that they understand each other. The best way to facilitate

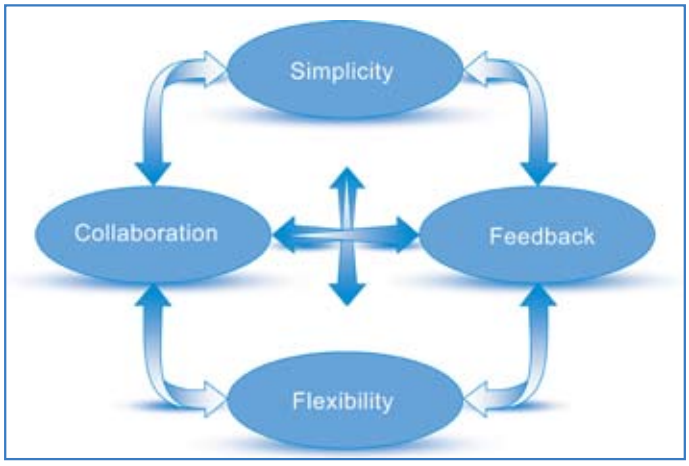


Figure 1: Simplicity as a core value in agile testing

this is by keeping things simple. Our test scripts should be easy to understand for both developers as well as domain experts. This allows them to review your work and provide you with important feedback. Fortunately, there is a good chance that your scripts are also more flexible when you keep them simple and easy to understand. As you might understand by now, simplicity, feedback, collaboration and flexibility are all interrelated and hard to separate.

Simplify test design and execution

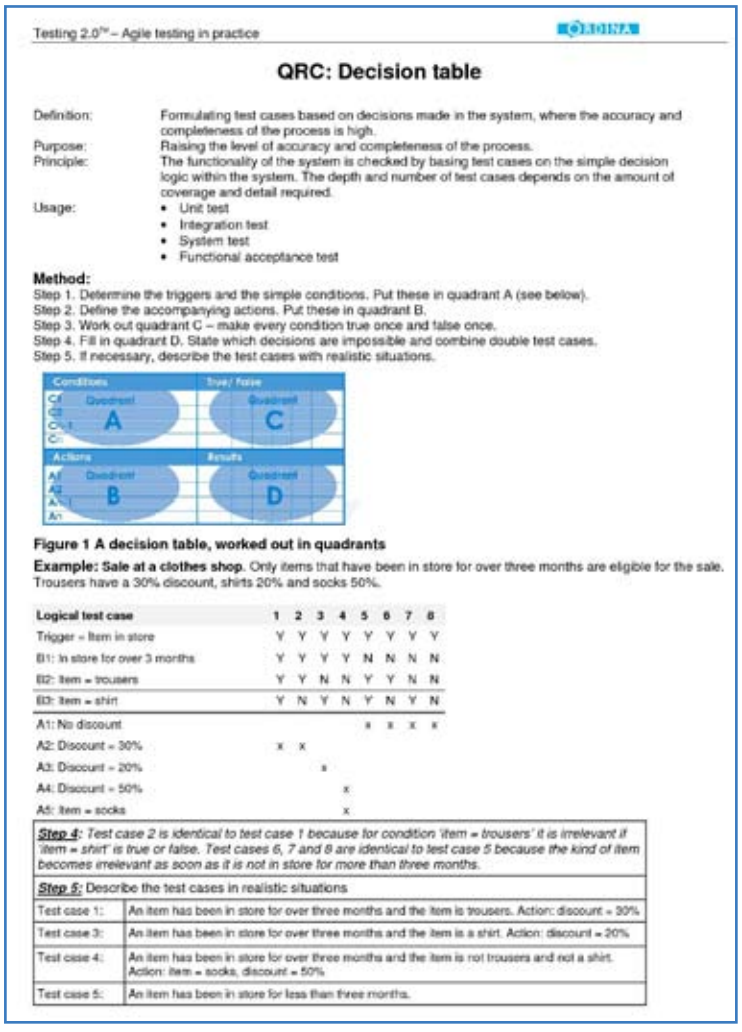
Testers seem to have a natural tendency towards complexity. We have learned formal techniques for deriving test cases from frozen specifications. Test scripts should be complete before test execution can commence. The truth is the exact opposite. The most useful feedback is generated not by deriving test cases, but by actually executing tests. Feedback is most valuable when it is given at the earliest possible moment. Hence, it is not desirable behavior for a tester to strive for completeness before executing any tests. Also, a tester learns a lot about the system when executing tests, allowing him or her to improve and add test cases along the way. Keep things simple at first and add complexity only when unavoidable.

The technique used to derive the test cases is not important most of the time. Selecting a heavy test specification technique might even give you a false sense of security. Creativity and insight are still the most important prerequisites for a successful test. The customer does not care what techniques you use as long as quality of the end product meets the standards. Again, keep things simple at first and add complexity only when unavoidable.

Quick-Reference Cards

Test specification techniques might be difficult to understand for developers, designers and domain experts. We believe that we can lift the overall quality to a higher level if we can transfer some of our knowledge of test techniques to our team members. This means that we need to make the techniques easier to understand for non-testers. We achieve this goal by summarizing the essence of a test technique into a single-page format, complete with an example. Such simple tools enable us to share knowledge about (agile) testing with other team members.

Besides test specification techniques, we also use the so-called Quick Reference Cards for other (agile) topics. As we believe in sharing knowledge, those QRC's can be downloaded at www.ordina.com .



Testing 2.0™ – Agile testing in practice

QRC: Decision table

Definition: Formulating test cases based on decisions made in the system, where the accuracy and completeness of the process is high.
Purpose: Raising the level of accuracy and completeness of the process.
Principle: The functionality of the system is checked by basing test cases on the simple decision logic within the system. The depth and number of test cases depends on the amount of coverage and detail required.
Usage:
- Unit test
- Integration test
- System test
- Functional acceptance test

Method:
Step 1. Determine the triggers and the simple conditions. Put these in quadrant A (see below).
Step 2. Define the accompanying actions. Put these in quadrant B.
Step 3. Work out quadrant C – make every condition true once and false once.
Step 4. Fill in quadrant D. State which decisions are impossible and combine double test cases.
Step 5. If necessary, describe the test cases with realistic situations.

Figure 1 A decision table, worked out in quadrants

Example: Sale at a clothes shop. Only items that have been in store for over three months are eligible for the sale. Trousers have a 30% discount, shirts 20% and socks 50%.

Logical test case	1	2	3	4	5	6	7	8
Trigger = Item in store	Y	Y	Y	Y	Y	Y	Y	Y
B1: In store for over 3 months	Y	Y	Y	Y	N	N	N	N
B2: Item = trousers	Y	Y	N	N	Y	Y	N	N
B3: Item = shirt	Y	N	Y	N	Y	N	Y	N
A1: No discount					x	x	x	x
A2: Discount = 30%	x	x						
A3: Discount = 20%			x					
A4: Discount = 50%				x				
A5: Item = socks				x				

Step 4: *Test case 2 is identical to test case 1 because for condition 'item = trousers' it is irrelevant if 'item = shirt' is true or false. Test cases 6, 7 and 8 are identical to test case 5 because the kind of item becomes irrelevant as soon as it is not in store for more than three months.*

Step 5: Describe the test cases in realistic situations

Test case 1:	An item has been in store for over three months and the item is trousers. Action: discount = 30%
Test case 3:	An item has been in store for over three months and the item is a shirt. Action: discount = 20%
Test case 4:	An item has been in store for over three months and the item is not trousers and not a shirt. Action: item = socks, discount = 50%
Test case 5:	An item has been in store for less than three months.

Figure 2: a Quick-Reference Card

Breaking down the black-box

Besides simplifying the way we prepare and execute our test cases, we can also simplify testing by breaking down the test object into smaller pieces. Sometimes it seems as if testers prefer looking at an information system as being one large black-box. We can understand why. We have been trained to solve black-box puzzles using generic approaches. But we can also do that on a lower level by breaking down the large black-box into smaller black-boxes. In the example we mentioned earlier, the calculation module is being treated as a small black-box inside a larger black-box. Doing so allows us to focus our testing effort on the areas that really matter, while at the same time we decrease the complexity by dividing the system into smaller comprehendible parts. This makes testing simpler.

Another advantage is that you can start testing important functionality without the need of a fully functional system. This gives you a head start, and you are able to provide early feedback not only for the development team, but also for the business. It's even possible to set up and perform acceptance tests on isolated parts of the system by involving domain experts and key-users.

Some say that opening the lid is against the rules of black-box testing. We don't agree. Black-box testing is all about testing based on input and expected output without the need to know how the results are produced. It's not that you are not allowed to know what's inside!

It's not unheard of to break down the walls even further, and be involved in white-box testing as well. Try pairing with a developer and write an important unit test together. Just remember, simplicity is the key to success.

Conclusion

Testing a complex information system can be quite a challenge. By involving developers and domain experts, we are able to tackle the complexity as a team. This allows us to work out creative and efficient test approaches. In traditional environments it might be harder to get your team members involved in testing. Our advice is to keep an open mind and always maintain a pro-active attitude towards your team members. Make sure your added value is visible and valuable to the team.

Working together as a team requires a certain amount of simplicity in everything you do, especially when people have different backgrounds. Keeping things simple sounds pretty easy and straightforward, but in fact most of the time it is quite hard. It takes a lot of experience to be able to transform something complex into something simple, but it's definitely worth striving for. We would like to end this article with a quote from Jazz bassist Charles Mingus. He had the following to say about simplicity:

"Making the simple complicated is commonplace; making the complicated simple, awesomely simple, that's creativity."

Biography

Jonne Ratterman:
Jonne is a test engineer at Ordina, based in the Netherlands. He has a master degree in Information Science and started his career as a software tester in 2005. He soon felt attracted to agile development, because of his interests and expertise in the areas of technology and management science. Jonne enjoys building bridges between testers, developers and the business in both traditional and agile environments.

Eric Jimmink:
Eric is a test consultant at Ordina. Having started as a developer, Eric shifted his focus to the testing field around 1998. He has been a practitioner and a strong advocate of agile development and testing since 2001.
Eric is co-author of 'Testen2.0 – de praktijk van agile testen' (Testing2.0 – agile testing in practice). Other publications include an article for Testing Experience that was reprinted in Agile Record, and several articles in Dutch ICT Trade magazines.

WHY DON'T YOU BE A SPEAKER AT INTERNATIONAL TESTISTANBUL CONFERENCE

TEST ISTANBUL

www.testistanbul.org

Software Testing: Quality Bridge Between Business and IT

Istanbul May 6, 7 - 2010

SEND YOUR PAPERS ON THE FOLLOWING SUBJECTS TO : www.testistanbul.org/openconf/openconf.php

SUBJECTS

Business Oriented Contents

- Business Analyst vs Test Specialist
- Business Knowledge in Software Testing
- Test for Business Needs
- Testing Outsourcing

Technology Oriented Contents

- Test Management
- Defect Management
- Test Automation
- Software Verification and Validation
- Testing Methods

Important Dates for Paper Submissions

Submission of papers until
February 28, 2010

Notification of acceptance
March 15, 2010

Submission of final paper
March 30, 2010

Submission of paper slides
March 30, 2010

TTB Turkish Testing Board
www.turkishtestingboard.org

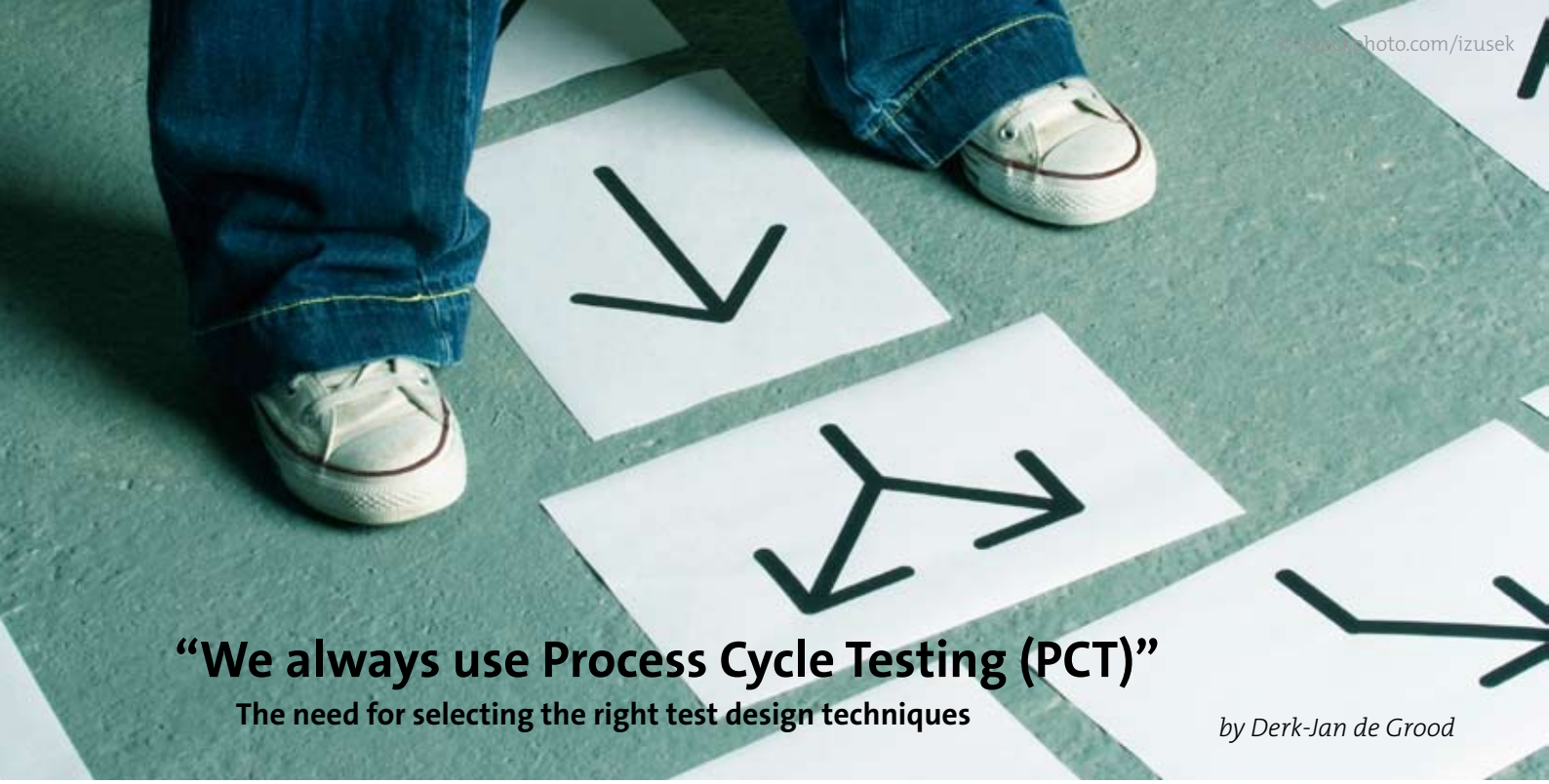
FOR DETAILED INFORMATION PLEASE CONTACT:

Tel: + 90 212 290 76 62, Email: info@testistanbul.org

www.testistanbul.org

Organized By Turkish Testing Board (www.turkishtestingboard.org)

ISTQB International Software Testing Qualifications Board
www.istqb.org



© istockphoto.com/izusek

"We always use Process Cycle Testing (PCT)"

The need for selecting the right test design techniques

by Derk-Jan de Grood

I once introduced test design techniques to my test department. We trained the testers and made "quick reference cards", explaining how each technique was to be used. Still, my testers were clearly struggling with the application of the techniques. When I sat down with one of the testers, it became clear why. In this specific case, my colleague derived test cases for the main scenario of a use case. He used a decision table. Each of the negative conditions in the decision table led him to the alternatives and exceptions that had already been defined in the remainder of the use case. I explained him that this way the technique was a useful review technique that could help to check the completeness of the use case. It was, however, not effective for defining test cases. To cut a long story short, he tried to apply the wrong technique on the wrong part of the specification at the wrong time.

We testers put a lot of emphasis on using test design techniques. Just to give an indication, TMap Next [TMap,2006] dedicates about 130 pages to test design techniques. This is about 17% of the whole book. "Foundations of software testing" [ISTQB, 2007] by Graham et al., dedicates about 20% of its content to explaining the functioning of different techniques. Syntax testing, Boundary Value Analysis (BVA) and State Transition Testing (STT) are typical examples. But there are many more techniques to aid the tester. Strangely enough, in most books, how to select a technique is hardly discussed. In my opinion, that is a shame, because selecting the right test technique is important for the efficiency of testing. My experience is that many testers find it difficult to select the right techniques.

Solid brick wall

Although we may have difficulties with the selection process, we generally agree that test techniques have great advantages. This is regardless of the techniques being used in a formal scripted way, or in the tester's mind while doing exploratory testing. Since each test design technique has its own focus, applying that technique will increase the probability that errors of a certain type will be found. The structural approach of the technique ensures that the work is done more thoroughly and that less obvious situations will be tested. It will therefore gain a more thorough understanding of the quality of the system and most likely result in errors found that would have remained unnoticed otherwise.

I recall a discussion that I once had with a test manager at a large insurance company. When addressing the problem of selecting the right techniques, she smiled and presented me with her solution; "We always use PCT". She had no problems with making a selection at all. This was a disturbing situation to me, since there is not one technique that is always the most effective. Unfortunately, the situation at that specific company is not unique. Many organizations describe a limited set of techniques in their test strategy, and these are used over and over again. I have heard of testers who tried to use different techniques for good reasons, and ran into solid brick walls.

Testing contributes to better product quality and trust in the system. As testers we should be well aware of the goals that the organization has with the product. What are the business drivers to which it contributes? What problems does it help solving? The testing process starts with learning about the goals of our stakeholders. Risk analysis is often used as a method to identify the threats to the anticipated goals and to understand what types of errors the business does not want to encounter during daily operations. Risk analysis identifies what information is relevant to the stakeholders. Test techniques can be applied in order to translate the risks, threats and unwanted errors, that have been determined, into well argumented test actions. By using the right techniques, the tester executes the right tests and is accountable for his actions and the quality of his advice. Since the techniques define the tests that are executed, they also determine the information that can be given to the stakeholders.

Stakeholder attention during triage meetings

Selecting the techniques to be used is a careful process. Selecting the wrong technique may cost you a lot of time without finding many errors or useful information. Failing to select the right technique may lead to major defects not being found (see textbox). It is important to link the techniques with the risk they cover and the type of errors they help to find. In order to increase this awareness, I often show bug reports in my training sessions. The attendees are asked to name test design techniques that could have been used to find the error. To understand this has a great advantage. If you can predict from experience that a certain bug report will not get a high priority and is likely to be put aside, do not use the techniques that will help you find these types of errors. You may know what kind of bug reports will cause commotion in the organization and get stakeholder attention during triage meetings. Select design techniques that will help you find those errors.

Test design technique effectiveness

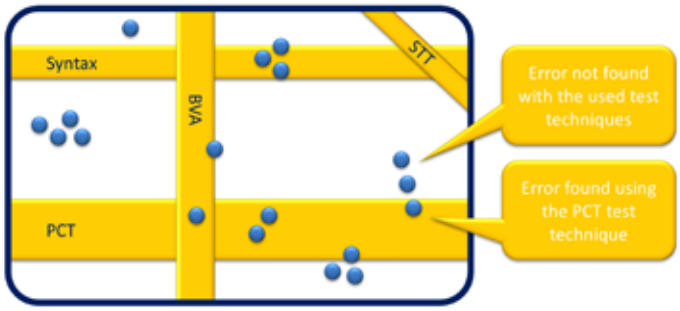


Figure 1
In the above figure, the test object is represented by the outer frame. The test object contains a number of errors, but we do not know where they are. If we did, we would not have to test, would we? The bands indicate the part of the code that is covered with test cases that were derived using test techniques. In this example four techniques are used, Syntax testing, Boundary Value Analysis (BVA), Process Cycle Testing (PCT) and State Transition Testing (STT). While executing the defined test cases we stumble upon several errors. From the example we can learn that

- Errors that are found by one technique are not necessarily found by another technique.
- Some techniques are very suitable in a specific situation and result in more errors found than others. In this example, using PCT has resulted in five errors, compared to STT none.
- Some errors cannot be found with any of the chosen techniques.

If you don't know what types of errors will catch your stakeholders attention, it's time to pay them a visit. Like we said before, typical risk analysis will lead to understanding this. The PRIMA method [SmarTEST, 2008], as an example, links the priority of the functions with 'types of errors'. This gives you precisely the information that you need; an indication of errors to look for and a measure of how important it is to find these.

With the PRIMA matrix, the first half of the puzzle is solved. It helps to answer the two questions: "What kind of errors do you want me to find?" and "What impact do these errors have in the production environment? ". The questions are included in the decision model that is depicted in figure 3.

Not because you can, but ...

Many times test techniques are applied because the tester is familiar with them and the test basis supports the technique. Using the decision model, we inverse the problem. We do not use techniques because we can, but because they help us to provide the right information. Unlike many methods we start with the errors and select the techniques that help to find them. In my book [TestGoal, 2008] I present a table that enables the tester to select the right techniques. The table below shows a few examples.

One other question needs to be answered. Is the required information available? We can choose a technique, but the required information needs to be available. You can decide to do state tran-

PRIMA product Risico Matrix — **Project: CRM implementatie**

Functions in the system

Types of Errors (Domains)

	100	24	7	9	17	12	13	5	12
	36	xxx	x		xx	x			x
	16	xxx	xxx	xx	x	xx	xxx		xxx
	6	xxx	x		x	xx			
	6	xx		xx	xx	x	xxx		
	6	xx	x	xx	x		xx		
	4	x			xx	xx			xxx
	4	xx		x	xxx				xx
	4	x	xx		xxx	x	xx	xxx	xx
	4		x		xx		xx		x
	4			x	xx	xx	xxx	xxx	

Figure 2

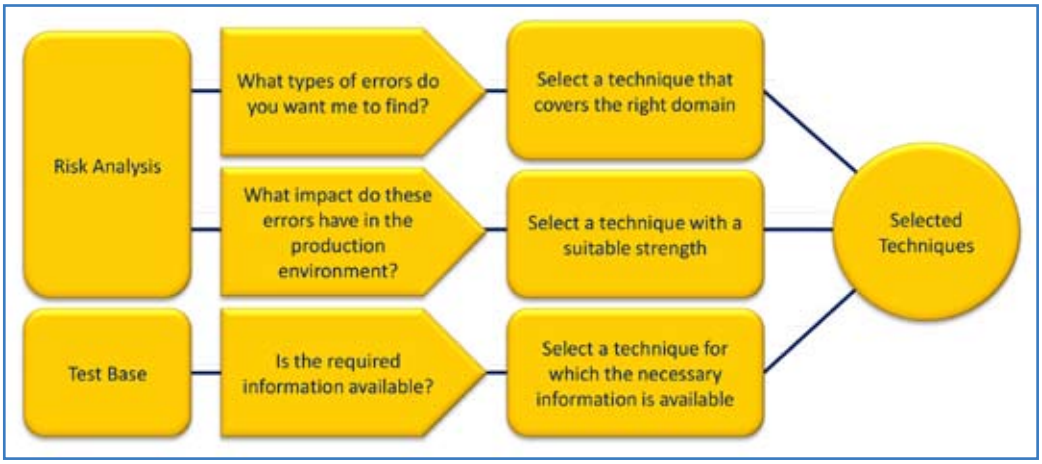


Figure 3

Errors	Technique	Required information
• Processing errors due to invalid inputs, e.g. invalid values, wrong units, etc. • Failing DB inserts due to invalid validation at GUI-message interface	• Syntax Testing	• Data dictionary explaining restrictions for each data element including conditional validation rules.
• Wrong functional decisions in for example business rules or recursive processing	• Equivalence Partitioning (EP) • Boundary Value Analysis (BVA) • Decision table testing (C/E)	• Defined decision points and conditions under which tasks are executed.
• Process breaks down and cannot be completed correctly • Process steps are taken in wrong order • Entities in the system can get into wrong state • System notifications are sent at the wrong moment or not at all • Unauthorized user is able to fulfill part of a process.	• Process Cycle Testing (PCT) • State Transition Testing (STT)	• Process description or swim lane diagrams • STD • Conditions under which certain transitions or process steps are taken • Authorization matrix

sition testing, but will you draw the necessary STDs if these are not provided? The table indicates a few examples of information that are required to apply the technique. There are two options when the test basis does not contain enough information:

- Find the information, no matter how hard it is
- Do not use the selected test design technique

The first option is preferable. Missing information or information that is hard to find is a risk. If you cannot find the information, other people involved in the project will probably have the same problem. Who knows which system behavior will be implemented, who will decide whether the application is good enough? Ensuring that the information becomes available reduces the risk of surprises arising later on in the software development project. But it's not always possible to obtain the missing information. In this case, the function will have to be tested in a different way, using less suitable or no techniques at all. It is important that this is communicated clearly; after all, some risks will not be addressed as intended and the information provided by the test project may be less precise and not take away all uncertainties.

Conclusion

Although there is much information available on test design techniques, very little is being said on how to select the right design techniques. The errors that you do not want to have in your production system should be the starting point when making your selection. Testers should therefore understand the relation between types of errors and techniques that help finding them. This understanding will lead to a better use of techniques and will lead to asking the right questions to the stakeholders.

Technique selection deserves more attention. I made a start by including it in my book. Who will take the challenge ? It would be a success if the selection of test techniques is explicitly addressed in the next test process that you'll implement, or in the next book that you'll publish. Any test manager or team manager that is ambitious to integrate the selection as part of the team's standard process is invited to mail me with his success or challenges.

References

- [TestGoal, 2008] TestGoal, result driven testing, Derk-Jan de Grood, Springer 2008, ISBN 9783540788287
- [SmarTEST, 2008] SmarTEST, Egbert Bouman, Academic services 2008, ISBN 9789012125970
- [TMap, 2006] TMap Next, Tim Koomen et. al., Tutein Nolthenius 2006 ISBN 9789072194794
- [ISTQB, 2007] Foundations of Software Testing, Dorothy Graham et.al., Thomson, 2007, ISBN 9781844803552

With many thanks to Egbert Bouman, Guido Dulos, Martin Kooij and Esther Martin.

Biography

Derk-Jan de Grood has broad, hands-on experience as test engineer, test manager and advisor in a large range of industries. As manager of various test departments he has learnt how to implement test methods the practical way. He gives lectures at various Dutch universities and is author of the first educational book specially written for teaching software testing at Dutch universities. Derk-Jan is also author of "TestGoal, the result-driven test philosophy".
As ISTQB advanced certified test manager and an ISEB accredited trainer, he provides training sessions on a regular basis. These training sessions vary from the standard introduction into result-driven testing to custom-made training courses that tune in on specific needs of the client. Besides that, he is a passionate, inspiring speaker at major testing conferences all over the world, such as the STAR conferences in the USA and Europe.

Recently, Derk-Jan has joined Valori. With that he started a new period in his career. He is in the program board of the 1st Dutch Test Conference. His new book on 'how to give a good presentation' has just been released and he started a column on testnieuws.nl.

Test Techniques - Why bother?

Column

by Erik van Veenendaal

Formal and informal test design techniques have been around for many years. Their uptake has, to say the least, not been the best. A recent survey revealed that a mere 50% of the test organizations actually use test design techniques in their projects, and only 25% used multiple test techniques, thereby being able to differentiate test coverage based on identified product risks. However, the recent growth in testing professionalism has brought many techniques to the foreground of the testing activity. There are many sources of information with respect to test techniques, their methods and coverage measure. (My favorite ones are listed at the bottom of this column). Let's look at a number of reasons why we should consider the use of test design techniques as part of our everyday testing activities, and who should be looking to use them. Test techniques come in many shapes and sizes, some formal and some not, some dynamic and some static. Almost inevitably, the focus will be drawn to the dynamic test design techniques and activities. However, static techniques must always be borne in mind, prevention will always be better than cure. Of course, there is not one complete answer to suit every situation, type of application or degree of quality required.

Are techniques for testers only?

Absolutely not! Everyone who is involved in product development, implementation and maintenance or use has a vested interest in establishing the quality of the product to be delivered. Specific techniques target one or more specific types of defect. Also, software engineers and business analysts should be taught techniques, including reviews, which reflect the development phase they are involved in. Everyone should be made aware of the type of tests they could carry out and the value they would add. If everyone in product development was aware of this issue and took the time to apply the basic aspects of test design techniques, then many, many, many defects would either not happen in the first place or be found before they could have an impact. The good news is that we at Improve Quality Services are providing more and more training and practical workshops regarding test design techniques to software engineers. Also, the current shift to agile development means testing is a team responsibility, and everyone should be involved in testing.

Why use testing techniques at all?

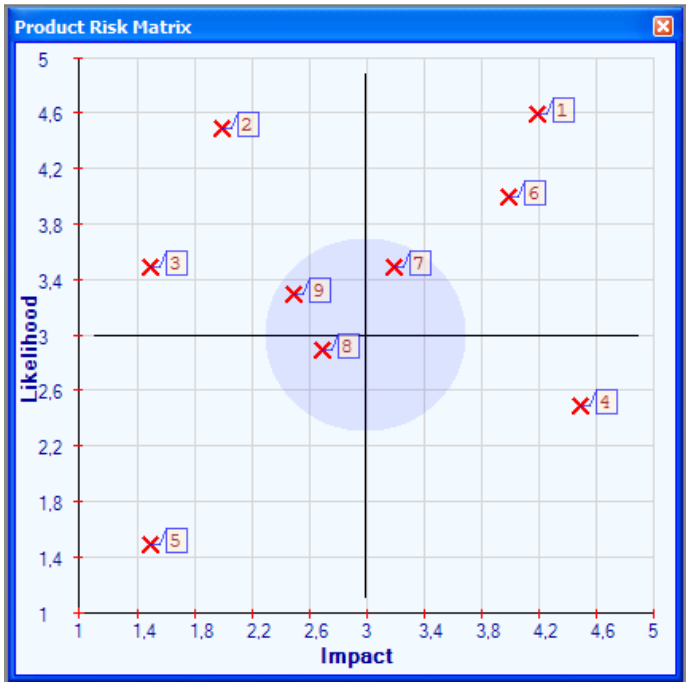
Why use test case design techniques at all? Maybe the following will convince you of their value:

- Objectivity; guarantees a certain level of coverage linked to an identifiable process of achieving it
- Increase in the defect finding capability (recent surveys indicate up to 25% more defects are found when testers are trained and use test design techniques)
- The ability to reproduce tests
- Building testware to support long-term system maintainability and test automation.

Test techniques provide an understanding of the complexities imposed by most systems. The use of techniques forces testers into thinking about what they test and why they are testing it. This is also for those who use exploratory testing; professional exploratory testers use the principles of test design techniques to draft their test ideas. In many cases, techniques identify a level of coverage that would otherwise be a mystery. Remember techniques will not provide a mechanism to exercise 100% test coverage. However, if information regarding the level of coverage to be achieved is available, then objective decisions can be made based on the risk of the system under test and on which tests to leave out.

Selecting techniques

The decision on which test design techniques are to be used and when is largely dependent on the understanding of the risk and in which component parts of the system it lies. Risk identification, assessment, management and mitigation are a fundamental part of the testing function. The identification and assessment of risk provides a focus to the testing, the management of risk is included as a part of the test process controls, and mitigation is achieved when the correct coverage of tests is performed. Often the test items are visualized in a risk matrix that identifies the most critical test items either from an impact (business) or likelihood (technical) perspective.



Example of a product risk matrix (from the PRISMA tool)

The choice of which test techniques to use also depends on a number of other factors, including the type of system, regulatory standards, customer or contractual requirements, test objective, documentation available, knowledge of the testers, time and budget, development life cycle, use case models and previous experience of types of defects found. Some techniques are more applicable to certain situations and test levels; others are applicable to all test levels.

Perhaps the single most important thing to understand is that the best testing technique is no single testing technique! Because each testing technique is good at finding one specific class of defect, using just one technique will help ensure that many (perhaps most, but not all) defects of that particular class are found. Unfortunately, it may also help to ensure that many defects of other classes are missed! Using a variety of techniques will therefore help ensure that a variety of defects are found, resulting in more effective testing.

Test modeling

It is readily agreed that by painting a picture or drawing a diagram thought processes can be explained to others, and perhaps

ourselves, more clearly. The vast majority of formal dynamic test techniques identified in BS7925-2 utilize one method or other to model the system component under test. This is not a coincidence. Modeling software is often considered the most complete method of establishing paths, decisions, branches and statements. The objective of formal test techniques is not just to provide an objective coverage measure, but also to document the decision making process, whilst making the component clear to both technical and non-technical staff. Testers must ensure they know what they are going to test from the possible options, and perhaps more importantly they must be able to identify which situations are not going to be executed. The level and detail of the modeling is dictated by the technique chosen. Testers must ensure they use these models to the best advantage to achieve a deeper understanding of how the system works.

Advantages / Disadvantages

So should testers use formal and informal test case design techniques? The decision to use or not use test case design techniques is directly related to the risk of the products being tested. Initially, a basic view of both the advantages and disadvantages must be taken [5].

Advantages	Disadvantages
Objectivity	Require training to some degree
Formal coverage measures	Time to implement – culture change
Early defect finding	Buy-in needed for everyone
Traceability	Not seen as useful for all types of application
Coverage independent of the tester	Takes more time than less formal test design
Way to differentiate test depth based on risks using different techniques	Does not cover all situations (error guessing will still be useful)
High level of re-use (re-usable testware)	Little use of domain knowledge of tester
Repeatability and reproduceability	
Audit trails	
Higher defect finding capability	

Whether testers like it or not, test case design techniques, whether formal or not, are an integral part of everyday testing life. If testing is to keep up in these ever-changing times, the testing discipline must ensure the validity of its function by continuing to be objective. In order to remain objective, the level of testing must be measurable. The only measure available is coverage, and only formal test case techniques and static testing methods provide this framework. If testing is to be taken seriously, the testing function must take itself seriously and ensure its objective is to make any testing objective. So should formal and informal test case design techniques be used? In one word: "YES".

Recommend literature

[1] Beizer, B. (1990), Software Testing Techniques, 2nd edition, Van Nostrand Reinhold, ISBN 1-850-32880-3

[2] BS 7925-2 (1998), Software Component Testing, British Standards Institution

[3] Copeland, L. (2003), A Practitioner's Guide to Software Test Design, Artech House Publishers, ISBN 1-58053-791-X

[4] Pol, M., R. Teunissen and E. van Veenendaal (2002), Software Testing; A Guide to the TMap approach, Addison-Wesley, ISBN 0-201-74571-2

[5] Veenendaal, E. van (2005), The Testing Practitioner, UTN-Publishing, ISBN 90-72194-65-9

Erik van Veenendaal is a leading international consultant and trainer, and recognized expert is the area of software testing and quality management. He is the director of Improve Quality Services BV. At EuroStar 1999, 2002 and 2005, he was awarded the best tutorial presentation. In 2007 he received the European Testing Excellence Award for his contribution to the testing profession over the years. He has been working as a test manager and consultant in software quality for almost 20 years.
He has written numerous papers and a number of books, including "The Testing Practitioner", "ISTQB Foundations of Software Testing" and "Testing according to TMap". Erik is also a former part-time senior lecturer at the Eindhoven University of Technology, the vice-president of the International Software Testing Qualifications Board and the vice chair of the TMMi Foundation.

Testing Clouds

No Need for Blue Sky Thinking

by Peter Cole

There is a new way of building and deploying applications around, and it has a variety of marketing terms to describe it, from SaaS to Virtualization, Cloud and even PaaS.

So what impact does this have on how we do our job as testers? The answer, as always, is that it depends. This article helps you to understand the differences, the challenges and what can be done to overcome them.

SaaS

Software as a Service (SaaS) is where a third party (e.g. Salesforce.com) hosts a standard application to all of its users. The application can be configured, and may offer scope for integration with some of the customers' own systems or data, but, fundamentally, the attraction of buying in SaaS is that someone else builds it, looks after it, updates it and tests it. The vast majority of the application lies outside your enterprise. For many enterprises then, there are two things to worry about.

Firstly, you need to test any custom configurations that you need to make the application suitable for your users. This means we can advance straight to end-to-end testing (or even user acceptance testing). This is because the changes made to the application are configured, they are not coded, the application leverages a library of common behaviors, which do not need detailed exhaustive testing, because they are already proven to work. This is the whole attraction and why we just concentrate on the end-user experience.

Secondly, you need to test your integrations. This can be the trickiest and most time-consuming part of the process. The integrations are obviously built by your organization and will often leverage a locally running component from the SaaS vendor which responds to events generated outside of your enterprise. Nowadays, increasing use is being made of Web Services (both SOAP and REST) to perform these types of integration, and thus any tool capability of generating this type of request can be used to test the integration points.

Cloud, Virtualization and PaaS

Cloud Computing, Virtualization or Platform-as-a-Service (PaaS) are all terms used to describe the principle of running parts (or all) of your application in a physical location and on physical hardware that does not belong to your company, accessed over the Internet (hence the term 'Cloud'). The key difference with Cloud is that the world has moved on, allowing you to use a machine instance for just a few minutes or for as long as you want, without worrying about buying it in the first place. Instead you just pay for the computer resources you use.

The approach required for testing these applications depends on their architecture. Applications can run entirely outside the enterprise, offering a user interface back to the users (often browser-based), or a hybrid approach where the application is composed of components, some of which run inside the enterprise, some of which run outside the enterprise in the cloud managed by the enterprise, and some provided by third parties running on their infrastructure. We're going to examine the hybrid class of application for the purposes of this article, trusting that you can adapt the approaches to suit your architecture.

One of the key benefits driving a particular subset of cloud-based applications is that of re-use. Organizations leverage services provided by other teams in their company, and by other companies to build a composite application. These composite applications increasingly rely on standards-based integration technology to facilitate their communication. SOA (often SOAP over HTTP or JMS), JINI, JavaSpaces, .Net and other technologies can all be used to create a loosely coupled application that can be easily distributed and even dynamically re-distributed across a Cloud. However, this is where the problems begin: testing any application, particularly once it is live, is made complicated by its dependencies. You may get away with rolling out phase 1 without a proper strategy, but by the time phase 2 comes along, you've got real problems.

The fundamental issue is that any useful testing is going to cause changes to happen to systems. In the interdependent, connected world of Cloud applications, at some point a real service is going to be called and cause a real side effect. You need a way to plug in simulations, or virtualized components, to replicate the behavior of the dependencies that are outside of your control. These components not only need to replicate the functional behavior of the system they replace, but also mimic the performance. Slow response times are used to simulate load. A flexible test automation environment will provide these for you, allowing you to concentrate on the business data of the simulation and delegate creation and execution of the virtualized component to the tool, decreasing development time and reducing maintenance costs. If the application is being tested in the Cloud, then most likely the virtualized components will need to run there, too. If the application is being tested inside the enterprise, then the virtualized components can run in either location.

Strong candidates for virtualization in the testing phase are also any services that have a pay-per-use model, or inflexible models such as requiring you to book in advance and keep to your slot. These models make it difficult for organizations to work in an agile fashion, causing delays and additional costs. Although cloud-based usage is chargeable, the fees are surprisingly reasonable compared to the cost of procuring and managing the hardware within the enterprise.

Of course, in the ideal world, these virtualized components are shared between development and testing teams, with the maintenance shared between them. In our experience, it is the testing team that ends up maintaining them and populating them with additional scenarios and business data.

Slowly Does It

The key to testing in these distributed environments is the same

as for SOA: the more components there are, the more places you need to look for problems. For this reason, each component should be introduced individually where possible. Until you are ready for the real component, replace it with a virtualized one to ensure that its dependants are functioning correctly.

The components themselves also require unit testing if they have been modified as part of this phase of work. If using test automation, you can always run the pack anyway to ensure there are no regression issues. Basic testing will have been carried out by the development team, but there is nothing like an experienced tester coming up with scenarios to fully exercise the component. This is an area where a new breed of test automation products is used. Testers can concentrate on building appropriate inputs and outputs for the test, generally at the business layer, rather than relying on how information is got into or out of the component. The automation product handles all of that for the user.

In projects where this virtualization approach has not been followed, introducing new components has simply caused the number of defects to increase out of control. The single biggest cause of problems in this area is poor shared understanding of what constitutes valid and invalid data traversing the interface.

Physics

Ultimately, all of these virtualized machines and components are going to run on physical machines. Architects and developers need to be wary of the actual effect on applications (e.g. many requests coming from a small number of IP addresses), and testers need to verify the application's architecture can handle this appropriately, rather than falsely designating such high volumes of traffic as a denial-of-service attack!

Nagging Doubts

Without a doubt, Cloud increases the complexity of the infrastructure, and this can make it hard to find an issue. When issues are hard to replicate, it is often easy to put them aside, after all, if it cannot be easily reproduced, it cannot be easily diagnosed, and thus cannot be easily fixed. This is obviously not good!

The problem for the Cloud-based application when automating end-to-end tests is that whilst the user interface may be visible inside the enterprise, at least some of the components are outside of the enterprise in the Cloud. Gaining access to these systems for the purposes of investigation is time-consuming and detracts from the testers' mission of finding, and then accurately reporting, defects.

The answer to this conundrum is to capture better information when the issue was discovered. A new generation of test automation products, of which Green Hat's GH Tester is one example, not only allows the automation of a test case for a Cloud-based system, but also the capture and verification of the side effects of executing the test. This allows touch points such as databases, log files and dependent interactions with other services to be captured and analyzed, offering the tester and developer valuable insight into the location of the defect and eliminating the need to reproduce.

Scaling at Runtime

Machines can dynamically scale up and down as the application runs, offering new problem areas not seen in previous application deployments as servers start up and close down again, and sessions potentially redistribute themselves. The impact of such changes on the running application is not something that needs to be verified every time, but does need to be checked against the architecture in its deployment environment.

Virtual Test Labs

Many of our customers have already introduced virtualization for testing environments inside their organizations, but Cloud offers a new possibility: hosting the testing environment outside of the organization. The cost advantages are obvious: only pay for the environment when you need it. However, the management of such environments is still an overhead with tools vendors rushing to catch up to facilitate the process of stopping and starting the environment, and deploying applications to it.

Emperor's New Clothes?

It is tempting to consider Cloud a brand-new way of doing things, but in fact it is incremental. The changes in behaviors required are not too dissimilar from those already required as applications have moved to component-based implementations and interdependencies over the past decade. Once the infrastructure and deployment model has been verified, there is no need to repeat those same tests for each project.

Summary

It is essential that some members of the modern testing team have an appreciation for application architecture and the application deployment environment and how this creates the need for new test cases. Dependencies between systems create problems once the system is live, and now - more than ever - it is important that systems are designed with testing in mind. New testing tools and an adaptation of traditional approaches can help teams work with the new complexities in the infrastructure. Cloud computing heavily emphasizes the need for effective testing at the design phase.

Biography

Peter Cole founded Green Hat in 1996 and today serves as President and CTO. Green Hat is committed to innovation in technology testing, and has delivered ground breaking advances in automated testing for complex SOA and BPM initiatives, including simulation. Peter works closely with other visionaries to understand the global trends in technology testing. He listens to global customers to understand how agile, cloud computing and other latest advances are affecting their businesses. Together with Green Hat's partners, Peter leads Green Hat to deliver award winning technology and processes that solve real business problems.

Peter holds a first class honors degree in Computer Systems Engineering from Warwick University, a leading UK research and teaching university. More information on Green Hat can be found at http://www.greenhat.com

testing experience

Knowledge Transfer

limited places

Rapid Software Testing

3-day tutorial with

Michael Bolton

March 29-31, 2010 in Berlin, Germany

Rapid testing is a complete methodology designed for today's testing, in which we're dealing with complex products, constant change, and turbulent schedules. It's an approach to testing that begins with developing personal skills and extends to the ultimate mission of software testing: *lighting the way of the project by evaluating the product*. The approach is consistent with and a follow-on to many of the concepts and principles introduced in the book *Lessons Learned in Software Testing: a Context-Driven Approach* by Kaner, Bach, and Pettichord.

The rapid approach isn't just testing with a speed or sense of urgency; it's mission-focused testing that eliminates unnecessary work, assures that important questions get asked and necessary work gets done, and constantly asks what testing can do to help speed the project as a whole.

One important tool of rapid testing is the discipline of exploratory testing—essentially a testing martial art. Exploratory testing combines test design, test execution, test result interpretation, and learning into a simultaneous, seamless process that reveals important information about the product and finds a lot of problems quickly.

www.testingexperience.com/knowledge_transfer.html

1400,- €
(plus VAT)

Guest Column

Test management 2.0

by Thomas Hijl

Isn't it time to get to version 2.0? That is what I have asked myself for a while ..... until a year ago. Now it is time to broadcast to the world that version 2.0 is available in beta. Here is a brief introduction.

Still many projects fail to meet stakeholders' expectations. Why is that? My opinion in general is that there is a lack of acceptance management. Lots of stakeholders, project managers and others are interested in test results, only because they manage their project result by numbers and/or colors. "More than 2 blocking issues is unacceptable," they say. However, after a while, a discussion starts about what is blocking and what isn't. Before you notice, they have reduced the severity to level 3. "So that problem is solved."

What they do not realize is that this might cause major problems in training sessions and during operation, or that there is a high risk that it may cause fatal damage to the project deliverable.

Management by numbers and colors will always cover up the real results of a project, no matter how hard the testers do their best to show the real facts. So, kick PowerPoint management out and introduce acceptance management instead. We should not be afraid of the real results. Even better,... we do not need to be afraid anymore, because we can manage acceptance.

Acceptance management

An acceptance manager like me will combine results from every project phase to an acceptable package for all stakeholders involved. How? By using two tools:

Involvement & Visualization

As an example (seen from a man's perspective): Imagine you wake up one morning and want to build a wooden construction in your garden for your kids. A place where they can play and enjoy themselves. With ropes, swings, a ladder and a small tree house with a roof. Beautiful. Ohhh,....how happy they will be.

It all started with a dream. After you have given it some thought, you tell your wife that you've had a good idea: "Let's build a wooden construction in the garden for our kids". Two things can happen:

1. She will hug and kiss you and tell you that you are the best man in the world, or...
2. She will tell you "Where did you get this crazy idea?".

It all depends on whether she accepts your idea. As us men know, it all comes down to charm and trust. These two need to be managed carefully to persuade your other half to get used to the idea and give you the go-ahead to actually start. Making her getting used to the idea, you keep coming back to the subject when you feel she is in a good mood. You give your thoughts more strength by making drawings of how it will be. You even show her pictures in magazines, where happy children play. When you have passed the level of acceptance of the idea, you have involved her with your visualizations. She is enthusiastic. Eureka!!

The next step is making plans and a concept. Phase 2 has started. Draw a plan, share it and come to an agreement that reflects the way you both want to go. The following weekend you are in your garden outfit, tools prepared and off you go. Look at yourself .You are building your dream.

.... I will spare you the rest of my experience with wooden constructions ;-) ....

The clue

Every project has the same stages and an important key to success is acceptance. With acceptance management you can enrich test management to get to level 2.0. Is that really true??? No, of course not. Acceptance management is another ball game. However, they can perfectly melt together and level project management to 2.0. It brings the balance we need so badly between 3 common project variables:

Money Time Quality

Test management can make the quality facts appear on the surface. Acceptance management can get these facts accepted. Project management can get it all done within the available time and budget.

Why project managers in general do not take this acceptance role? You might ask. Well,... it is nearly never part of their assignment. They need to get the project done within the given time and budget. Quality is hard to measure and most of the time not specified properly, so they will always cut down on that. The test manager always operates under the project manager so he/she has not enough strength to turn the tide. In the end it means "not my problem" for customers. They will keep getting low quality for a high price too late. Putting an acceptance manager in place that reports directly to the customer brings the balance back and we can make customers happy again with the project deliverables. The deliverables need to be at an acceptable level for the demand as well as the delivery side. That's the sole assignment of the acceptance manager in a continuous process.

The question that will keep coming back...is this: "Can we all accept the consequences of our decision?" And they all respond: ..... "YES, WE CAN!!!" .... I love it when a plan comes together ....

How to

A small peek in the acceptance tool-box of my working method:

- Follow the trends of automated testing, but keep acceptance testing independent and human.
- No need to involve testers in early stages of the project, use an acceptance professional instead.
- Accept every project stage by passing through a quality gate, where all stakeholders are involved.
- Visualize your acceptance progress and plans, so everybody can see we are really getting somewhere.

Thomas Hijl is acceptance manager and partner at Qwince BV. He started in 1996 as consultant-project manager for Philips Semiconductors (a.k.a. NXP) working on technology projects in the EMEA, APAC and AMEC regions. He switched to test management because of his ambition to improve project deliverables. Now, he serves clients with managing acceptance throughout a project lifecycle, facilitating all stakeholders in acceptance of common interests (by understanding).

Test 2.0 - Advanced Testing Techniques in Life Insurance

A New Quality in Mathematical Testing of Portfolio Management Systems in Life Insurance

by Jens Fricke & Thomas Niess

The demands on mathematical testing of portfolio management systems are increasing: more than ever before testing requires a structured and systematical approach. The reasons for this are ever-shortening product life cycles, annually changing surplus parameters, new legal regulations and increasingly complex products.

The testing procedures used in insurance up to now do work. They can, however, be optimized significantly by structured test processes according to the ISTQB testing process and standardized documentation.

Possible improvements are described below according to the V-model and the ISTQB testing process.

Mathematical Testing

The software component mathematics (mathematical calculation core, actuarial sub-system, calculation model, and so on) is the subject of mathematical testing of portfolio management systems in the insurance industry. It comprises all mathematical values of the portfolio management system.

These values emerge when processing a business transaction (information, technical change, update), in printed material (policies, updates of changes and dynamics), in interfaces (collection/payment interface, printing interface and so on) and in persistent values in the databases (see Figure 1). Thus mathematical testing delimits itself from technical testing (test of total functionality) and the testing of peripheral systems (collection/payment, commission).

Mathematical testing of portfolio management systems basically checks the quality characteristic functionality. It comprises all characteristics that describe the required capabilities of the system. The objective of the test is to prove that each required capability was realized in the system in such a way that the input/output behavior or a specified reaction is fulfilled. According to ISO standard 9126 the characteristic functionality consists of the sub-characteristics suitability, accuracy, interoperability, compliance and security.[1] Interoperability and security are not relevant for the testing of the software component Mathematics. They are the objectives of the technical testing. For mathematical testing the sub-characteristics suitability (each required capability exists in the system and was realized in a fitting way), accuracy (the system provides the correct or specified reactions or effects) and compliance (the software meets the generally acknowledged actuarial principles) are important.

Tests have to be executed in a structed way and systematically, because testing a software system can never be complete due to the multitude of possible combinations. This is the only way to apply limited resources to specific targets, to detect as many failures as possible with reasonable effort and, if possible, to avoid unnecessary testing (testing that does not lead to new findings). The procedures to derive test cases can ideally be combined with the experience of the testers and typical cases that occur in actual production.

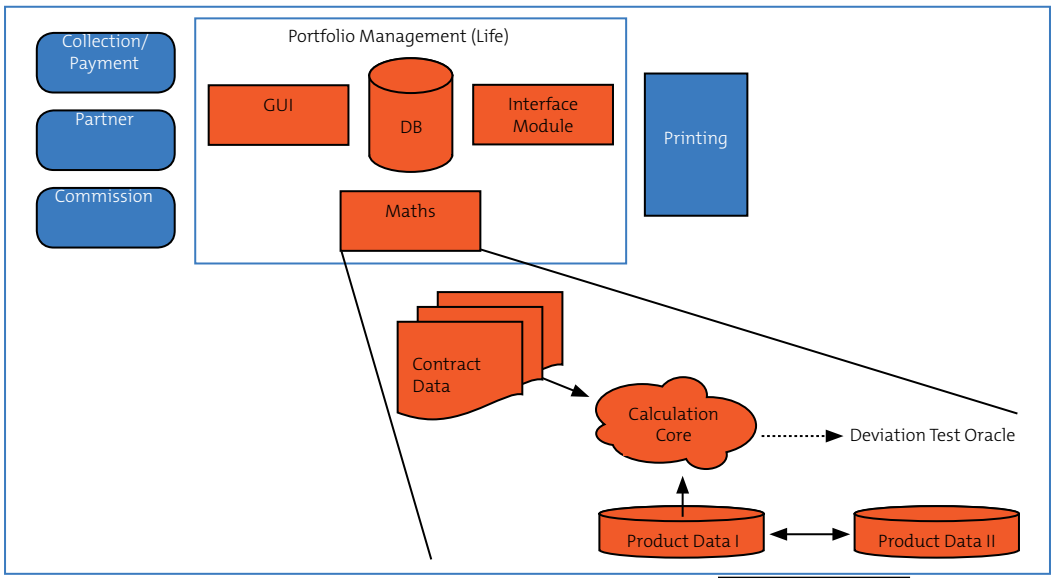


Figure 1: Mathematical Testing in the Context of a Portfolio Management System

1. Test Levels in Mathematical Testing during the Software Life Cycle

The V-model is very well suited to contemplate the test levels in the context of the development/maintenance of a portfolio management system. The levels relevant for mathematical testing are the steps com-

1 See ISO/IEC 9126

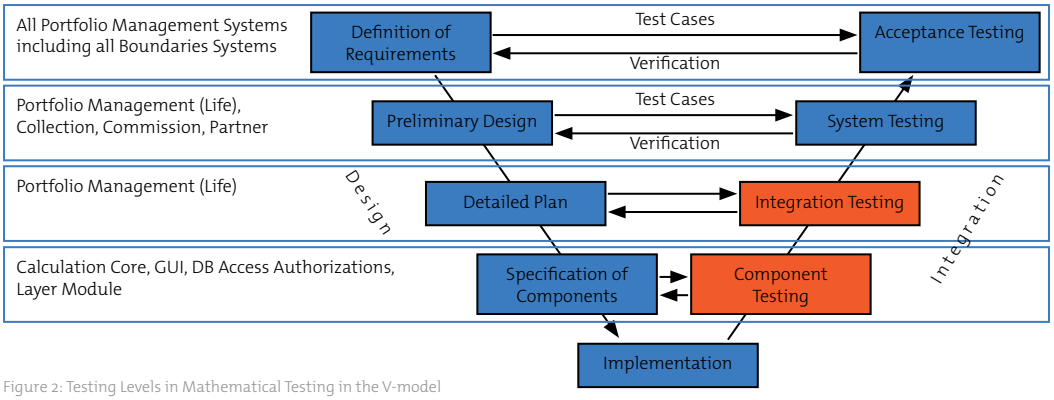


Figure 2: Testing Levels in Mathematical Testing in the V-model

ponent testing and integration testing. The other test levels - system testing and acceptance testing - play only a minor role. REF _Ref246816059 \h Figure 2 shows the steps of the V-model and their significance in the life insurance industry. The levels relevant for the mathematical test are indicated in orange.

1.a. Component Testing

The test level component testing subjects the partial system mathematics (calculation core, actuarial subsystem, and the like) as a whole to systematical testing for the first time. Characteristically, the separate software module mathematics is tested without the complete administration system. Thus influences from outside of the component are excluded. A detected failure can be attributed directly to the component tested.

Component testing is the first level in testing after implementation, therefore testing is closely connected to development in this level. Processing test cases without the complete administration system requires a test driver. The development of this driver requires the knowledge of developers. This is why component testing is often called developer testing, even though the tests are not executed by the developers themselves.

The most important objective of component testing is that the test object realises the functionality demanded in its specification correctly and completely (functional test). To check the correctness and completeness of the implementation, the component is subjected to a number of test cases. Typical software defects of functional component testing are calculation errors, missing program paths or the selection of wrong paths. Testing for robustness is another important aspect of component testing although it does not concern mathematical values. This test is executed in analogy to functional component testing. Test input consists of values that are invalid according to the specifications, and suitable exception handling is expected from the component. These negative tests are to prevent the complete system from crashing when an error occurs in one component.

In modern portfolio management systems products and tariffs usually are defined using numerous parameters (cost and return sets, formulas) called product data. The systematical test of these settings can also be assigned to component testing. Here, the component product data (product data definition system or the like) is subjected to static testing, i.e., data is tested without running the software.

A manual check of the product data requires a lot of time and effort. The results of the analysis usually have a lesser quality than results that were obtained automatically. This is caused both by the amount of the product data (products, tariffs, modules, surplus parameters) and by their complexity. Support by a suitable tool, called product data reconciliation, can be helpful. Typical tasks in product data reconciliation are finding out whether there are two similar products or tariffs (e.g., old and new tariff set), or whether two different product data versions (earlier development and current release) vary in specified changes only. In this tool-supported analysis two products or tariffs or two different versions of product data (e.g., before and after realisation of certain requirements) are compared automatically and the results are output in a suitable way. Next, it is checked whether these differences can be explained by the implemented requirements. The tasks mentioned can be automated using database-based tools. The testing instrument product data reconciliation can complement or even partially replace tests for new profit declarations or product generation.

Component testing can be executed in the form of white-box testing, because it is closely connected to development and the tester often has access to the source code. However, consisting of various complex components the mathematical component itself is too large to design test cases using white-box procedures (statement, branch or path coverage) with justifiable effort. This will only be possible for selected components. In practice mathematical component testing is almost exclusively executed in the form of black-box testing. The test coverage of the test cases specified this way can be measured using tools and complemented by further test cases for those parts of the code that have not been executed so far.

1.b. Integration Testing

In integration testing the modules verified in component testing are tested with regard to their interaction. Integration testing assumes that the test objects transferred (the separate components) were already tested and faults corrected. It is tested whether the interaction of the mathematical component with the remaining administration system works correctly. The objective of integration testing is to find defects in the interfaces and in the interaction of the mathematical component with the rest of the administration system.

In this test level tests can be executed on the complete administration system without peripheral systems for the first time. Therefore this test level also requires test drivers or dummies to provide test data from the peripheral systems (i.e. partners) for the administration system. Here, it has to be ensured that values verified in component testing are correctly processed in the complete portfolio management system, e.g. the way they are displayed on the interface GUI (see Figure 3).

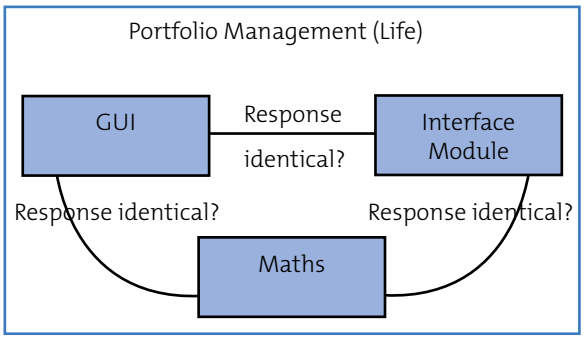


Figure 3: Tests in Integration Testing

Problems can already occur during integration. Problems that are more difficult to find concern the interaction of the components and can only be discovered in dynamic testing. They include, for example, missing or syntactically wrong data which result in a component not working or crashing. Wrong interpretation of transferred data or data transfer at an incorrect or late point in time are other possible problems. These faults manifest themselves in the interaction of the components and thus cannot be detected during component testing.

In practice it often happens that mathematical component testing is dispensed with, and instead all test cases are executed in

HANDBUCH

TESTEN
IN DER FINANZWELT

Das Qualitätsmanagement und die Software-Qualitätssicherung nehmen in Projekten der Finanzwelt einen sehr hohen Stellenwert ein, insbesondere vor dem Hintergrund der Komplexität der Produkte und Märkte, der regulatorischen Anforderungen, sowie daraus resultierender anspruchsvoller, vernetzter Prozesse und Systeme. Das vorliegende QS-Handbuch zum Testen in der Finanzwelt soll

- Testmanagern, Testanalysten und Testern sowie
- Projektmanagern, Qualitätsmanagern und IT-Managern

einen grundlegenden Einblick in die Software-Qualitätssicherung (Methoden & Verfahren) sowie entsprechende Literaturverweise bieten aber auch eine „Anleithilfe“ für die konkrete Umsetzung in der Finanzwelt sein. Dabei ist es unabhängig davon, ob der Leser aus dem Fachbereich oder aus der IT-Abteilung stammt. Dies geschieht vor allem mit Praxisbezug in den Ausführungen, der auf jahrelangen Erfahrungen des Autorenteams in der Finanzbranche beruht. Mit dem QSHandbuch sollen insbesondere folgende Ziele erreicht werden:

1. Sensibilisierung für den ganzheitlichen Software- Qualitätssicherungsansatz
2. Vermittlung der Grundlagen und Methoden des Testens sowie deren Quellen unter Würdigung der besonderen Anforderungen in Kreditinstituten im Rahmen des Selbststudiums
3. Bereitstellung von Vorbereitungsinformationen für das Training „Testing for Finance!“
4. Angebot der Wissensvertiefung anhand von Fallstudien
5. Einblick in spezielle Testverfahren und benachbarte Themen des Qualitätsmanagements

Herausgegeben von Norbert Bochynek und José M. Díaz Delgado

Die Autoren
Björn Lemke, Heiko Köppen, Jenny Siotka, Jobst Regul, Lisa Crispin, Lucia Garrido, Manu Cohen-Yashar, Mieke Gevers, Oliver Rupnow, Vipul Kocher

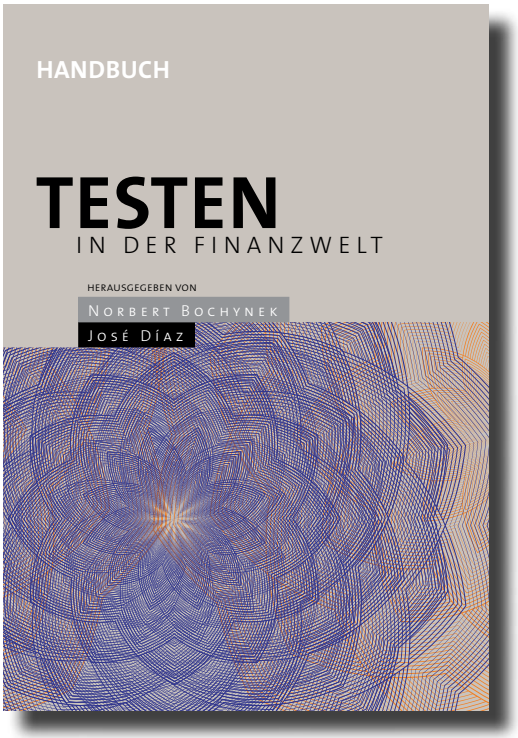
Gebundene Ausgabe: 431 Seiten

ISBN 978-3-00-028082-5

1. Auflage 2010 (Größe: 24 x 16,5 x 2,3 cm)

48,00 € (inkl. Mwst.)

www.diazhilterscheid.de



integration testing. The disadvantages in this are:

- Most of the failures detected are caused by functional defects in separate components. Implicit component testing is executed in an unsuitable test environment.
- Some failures cannot be provoked, because there is no access to the separate components, thus many defects cannot be detected.
- When a failure or malfunction occurs during testing, it can be difficult to isolate the point of origin and thus the cause.

The reverse case, foregoing integration testing and processing all test cases in mathematical module testing, also exists in practice. The disadvantages of this procedure have already been discussed above.

1.c. System Testing

System testing is of minor importance for mathematical testing. In it, it has to be examined whether the values of the peripheral systems (like printing, provision or monetary transactions), that were verified in the previous test levels, are processed correctly. The testing of mathematical values in this test level is limited to comparing whether the correct data were accessed, because the values used here were already verified in component testing or integration testing (see Figure 4).

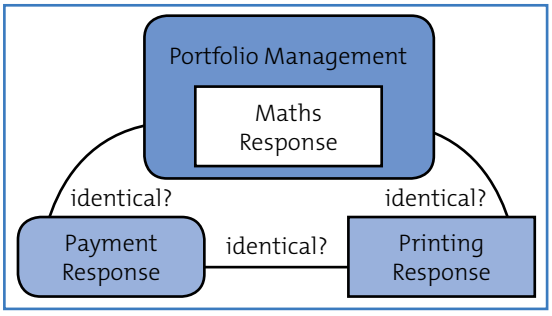


Figure 4: Test in System Testing

An exception is testing in billing and financial accounting. This testing can only be executed in test level system testing because it requires programs for further processing (outside of the portfolio management system).

1.d. Acceptance Testing

In acceptance testing the complete (possibly cross-divisional) IT environment is tested for acceptance. In this test level mathematical testing is not relevant anymore. The tests cases relevant for acceptance testing were already executed in the previous test levels.

1.e. Regression Testing

Regression testing is to ensure that modifications in the portfolio management (enhancement, correction of defects) do not have any undesired effects on the functionality. Here test cases are processed in different versions of the portfolio management system, and the output of the systems is compared. The evaluation whether a test case was successful is not based on a comparison with the specifications but on the comparison of the newly created output to that of the previous version or the output that up to now was regarded as correct. If there are no differences, the regression test case was successful. In case of differences, these have to be analysed. If the differences are wanted, the changed behaviour has to be defined as target behaviour (reference case) for future regression testing. If there are differences that are not desired, the defect has to be localised and corrected.

Regression testing can be executed as a diversifying test[2], i.e. against a previous version, or as a function-oriented test (against references, i.e. the specifications). Testing against references is to be preferred to diversifying testing, because it can reduce the number of defects creeping in. For the analysis of deviations, diversifying regression testing can be a helpful addition. It can clarify whether a deviation occurred for the first time in the current release or just was not detected up to now.

Regression testing lends itself in particular for mathematical component testing. Furthermore, regression testing of the complete portfolio management system and a comparison of mathematical values make sense. This requires a close coordination between mathematical and technical testing.

Regression testing is an indispensable part of the development and maintenance of a portfolio management system, because portfolio management systems are permanently being enhanced (new functionalities, new product, legal requirements).

2. The Fundamental Test Process

Figure 5 shows the testing process[3] developed by the ISTQB (International Software Testing Qualification Board) and relates the separate activities to the test documents according to standard IEEE 829[4].

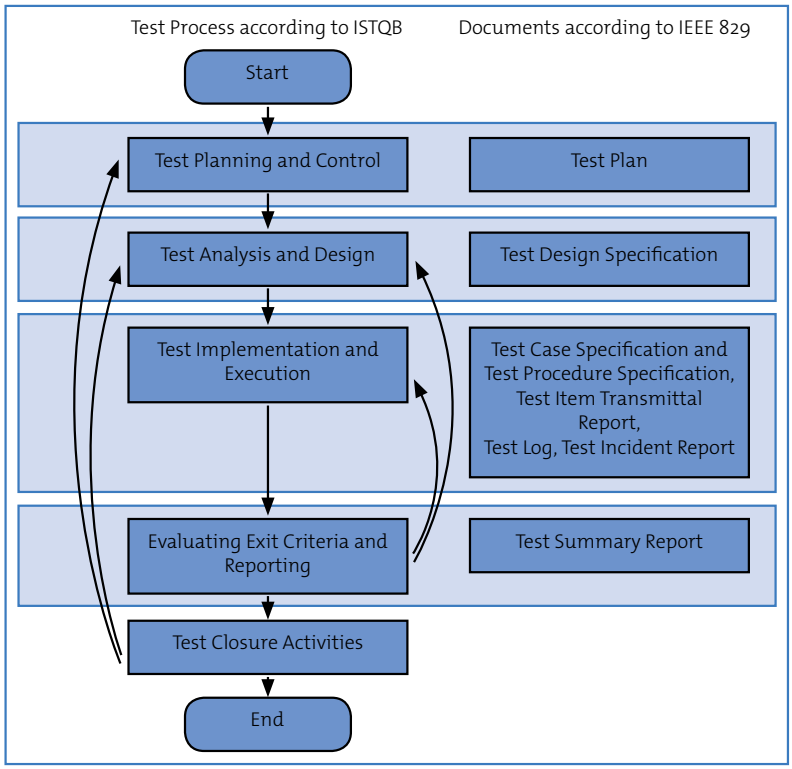


Figure 5: Test Process according to ISTQB and Corresponding Documents according to IEEE 829

2.a. Test Planning and Control

In test planning and control the project manager has to clarify and record what has to be tested, what the objectives of the tests are, when testing is to be started, where testing is to be executed, which personnel is to execute the tests, and so on. In his planning the project manager usually does not go into the details of the requirements. In most cases, the level of details is such that specific requirements or subjects are assigned to each tester. The project manager records the results in the test plan. Test controlling can be conducted according to this document. No peculiarities for mathematical testing result from this phase of the test process.

2 See Peter Liggesmeyer: Software-Qualität - Testen, Analysieren und Verifizieren von Software. Spektrum Akademischer Verlag, Heidelberg/Berlin 2002

3 See Andreas Spillner und Tilo Linz, Basiswissen Softwaretest: Aus- und Weiterbildung zum Certified Tester - Foundation Level nach ISTQB-Standard, Dpunkt Verlag, August 2005.

4 See IEEE 829

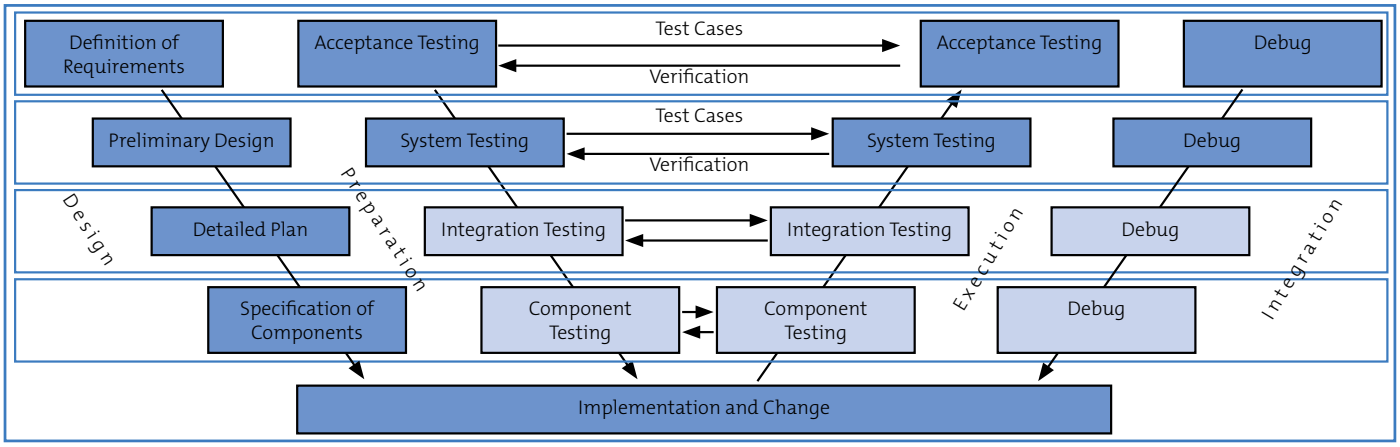


Figure 6: Levels of Mathematical Testing in the W-Model

2.b. Test Analysis and Design

In test analysis the basis for testing (requirements, specifications) is reviewed and analysed with regard to a sufficient level of detail for the creation of test cases. If necessary, the originator has to be consulted or amendments have to be made. Here, the testability is evaluated based on test basis and test object. Test specification is conducted according to the W-model (see Figure 6) if possible in parallel to the drawing up of requirements. This permits the early quality assurance of the requirements.

Once the analysis has been completed, logical test cases are defined (also called test design). Test cases can be specified based on the specification of test objects (black-box procedure) or on the program code (white-box procedure).

In mathematical module testing (component testing), white-box procedures like statement, branch or path coverage can be used, as can black-box procedures like equivalence partitioning. Testing level integration testing uses black-box procedures only. Use of white-box procedures does not make sense, because testers usually do not have access to the program code.

The creation of logical test cases – and the recording of the test design specifications specified according to IEEE 829 – is the first of two steps in the specification of test cases. The second step is part of the test process phase test implementation and documentation, which will be described later.

Various types or methods of testing can be applied when creating logical test cases. Among them are functional testing, which tests the visible input/output behaviour of the test object, structure-based testing, in which the internal structure of the test object is used for testing, and change-based testing, i.e., regression testing after changes to the system. Structure-based testing in a compound component like the mathematical sub-system requires a lot of effort, therefore requirement-based, functional testing is applied in both mathematical component testing and mathematical integration testing.

The requirements to be implemented are used as basis for testing. For each (new) functionality described in the requirements, at least one test case is derived. Test cases can be further refined using additional black-box testing methods like the boundary value analysis. If, based on experience in other projects, a tester assumes that a defect may occur with a particular combination, this test case should be included in the selection of test cases. This is also known as the experience-based approach.

In practice there also are requirements concerning certain subjects, in which the functionalities cannot be read directly from the requirements. Examples for this are new tariff generations or new profit sets. Here it is assumed that the already existing functionalities are valid for these requirements, too. Test cases have to be created carefully using equivalence partitioning. The greatest danger in this is defining too many test cases. To reduce the required effort, selected test cases can be complemented by regression testing and by checking the production data.

When defining the expected results (test oracle), target data is derived from the input data using the specifications. In mathematical testing of portfolio management systems, test tools are often used as test oracles (back-to-back test). In this, a test tool (Excel, VBA, APL, or the like) is programmed according to the specifications used in the implementation of the portfolio management to compare the mathematical values of the portfolio management system. In case of deviations in the results from both versions after processing of the same input data, these deviations are analysed and the causes are resolved. In case of identical output, the test result is regarded as being correct. Identical defects in the diverse versions cannot be detected using this testing method.

The offer program can also be used as test oracle, because it has to be programmed anyway. It is, however, of limited use only, because usually programs to create offers can only test the issuing of policies for current tariff sets. Technical changes, inclusion/exclusion of tariffs and old tariff sets cannot be tested this way. Often a program to create offers uses the same calculation core as the portfolio management. In this case the use as test oracle is not an option. Manual calculation of the mathematical values of a portfolio management system from the specification often requires a lot of effort. Here, back-to-back testing provides a simple option to reduce the time required.

The test oracle (testing tool) should be realised early on during test design. A precise definition of the expected values is not necessary when using testing tools as test oracles. It is sufficient to complete the development of the tool before the test is executed ("post-calculation is easier than pre-calculation"). There are, however, cases in which it is very helpful for the developer to know the expected results. Most importantly this is the case when correcting test incident reports (defects reported by defect tracking).

Criteria for the successful processing of test cases are also part of the definition of logical test cases. A suitable metric to determine the classification level is, for example, the deviation between the calculated and expected value of an output parameter. These deviations can be percentages or absolute values. A possible level of classification in regard to absolute differences for acceptance is:

- < 1 Cent: acceptable
- 1 Cent – 1 EUR: conditionally acceptable. It has to be analysed whether the deviation is, e.g., a deviation between test oracle and portfolio management system due to rounding (e.g. rounding the insurance sum to EUR) or a systematic deviation.
- > 1 EUR: not acceptable

2.c. Test Implementation and Execution

Test implementation consists of preparatory tasks for the test. When the test process has proceeded and there is more clarity about the technical implementation, the logical test cases are

limited places

testing experience

Knowledge Transfer

Hands-on Agile Acceptance Testing and Specification by Example

3-day tutorial with

Gojko Adzic

April 19-21, 2010 in Berlin, Germany

This three day workshop immerses the participants into a project driven by Specification by Example and Agile Acceptance Testing. Through facilitated exercises and discussion, you will learn how to bridge the communication gap between stakeholders and implementation teams, build quality into software from the start, design, develop and deliver systems fit for purpose.

This workshop is aimed at testers, business analysts and developers. It combines an introduction to Specification by Example and Agile Acceptance Testing, a set of exercises to help you get started with FitNesse - the most popular tool for agile acceptance testing - and a full day of working on realistic domain examples taken from your recent projects or a future phases of projects. This ensures that you gain real-world experience, enabling you to kick-starting internal adoption of these practices in your team.

www.testingexperience.com/knowledge_transfer.html

1250,- €

detailed and recorded in the test case specifications defined according to IEEE 829. These detailed test cases can then be used in testing without further changes or amendments if the defined preconditions are valid for the particular test case. The detailed course of testing is to be defined in the test procedure specification (see IEEE 829). In this the priority of the test cases has to be considered, and test cases have to be grouped in test sequences (also called test suites) and test scenarios (also called test scripts) to be able to effectively process the test cases and to get a clear structure of the test cases. In case a test has to be terminated prematurely, prioritising is to ensure the best possible result up to that point in time. For mathematical testing of a new tariff set, for example, issue of policy and update can be tested with a higher priority than technical changes.

Before starting the test, test beds have to be programmed and implemented in the environment concerned. The correct build of the test environment must be checked because failures can be caused by error conditions in the test bed. Also, the test oracles (testing tools) have to be programmed and must be executable.

After completing the preparatory tasks for the test, testing can be executed immediately once the parts of the system to be tested have been programmed and transferred. Testing can be executed manually or with tool support, and the predefined sequences and scenarios executed. In module testing, test beds and test drivers often allow to automate the tests.

First, it is checked whether the parts to be tested are complete. This is done by installing a test object in the available test environment and testing for general starting and execution capability. It is recommended to start with testing the main functionalities of the test object (smoke test). In case failures or deviations from the target results occur here already, it makes little sense to go deeper into testing.

While executing the test, the test cases processed and the results achieved (successful or with faults) have to be documented in the test log (see IEEE 829). Based on the test logs, the execution of the test must be traceable for persons that are not directly involved. It must be possible to verify that the planned test strategy was realized and to see which parts were tested when, by whom, in which detail and with which result.

Besides the test object, test beds, input files and test logs are part of each testing procedure. This information has to be managed in such a way that it is possible to repeat the test using the same data and frame conditions at a later point in time without any problems (configuration management of testware).

In case of a difference between actual and expected results during testing, the deviation has to be analysed, and it has to be determined whether there really is a failure, in which case a first check of possible causes has to be conducted. For this it may be necessary to specify and run additional test cases. In case deviations can be traced back to failures, this is documented in a test incident report (see IEEE 829). The causes for discrepancies to the target result can also be a faulty or inaccurate test specification, a faulty test infrastructure, a faulty test oracle (especially in second programming in testing tools) or a faulty test case or test run. Deviations have to be checked and analysed most carefully. A reported alleged failure which was caused by a faulty test case or faulty programming in the test oracle can damage the credibility of the tester. This must, however, not lead to potential failures not being reported just to be on the safe side. When documenting the deviations in a test incident report, it is important to ensure that a third person can understand the failure. The desired behaviour and expected values have to be indicated. In contrast to test design, in defect tracking it is not sufficient to indicate the testing tool used. The developer who has to correct the defect cannot be expected to deal with the testing tools.

Measurements of the testing coverage have to be conducted to monitor progress. This task is the responsibility of test management, because the individual testers will aim to execute all test cases.

Once the defect has been corrected, it must be checked whether the failure was removed and no new defects were added during the correction (regression testing). If necessary, new test cases have to be specified to check the program modules that were changed.

2.d. Evaluating Exit Criteria and Reporting

This phase evaluates whether the test completion criteria defined in planning have been met. This activity does not put special demands on mathematical testing. The tester should, however, support the project manager in clarifying discrepancies.

2.e. Test Closure Activities

In this phase, the experiences gained in the previous phases are analysed and the lessons learned are documented. This activity also does not put special demands on mathematical testing.

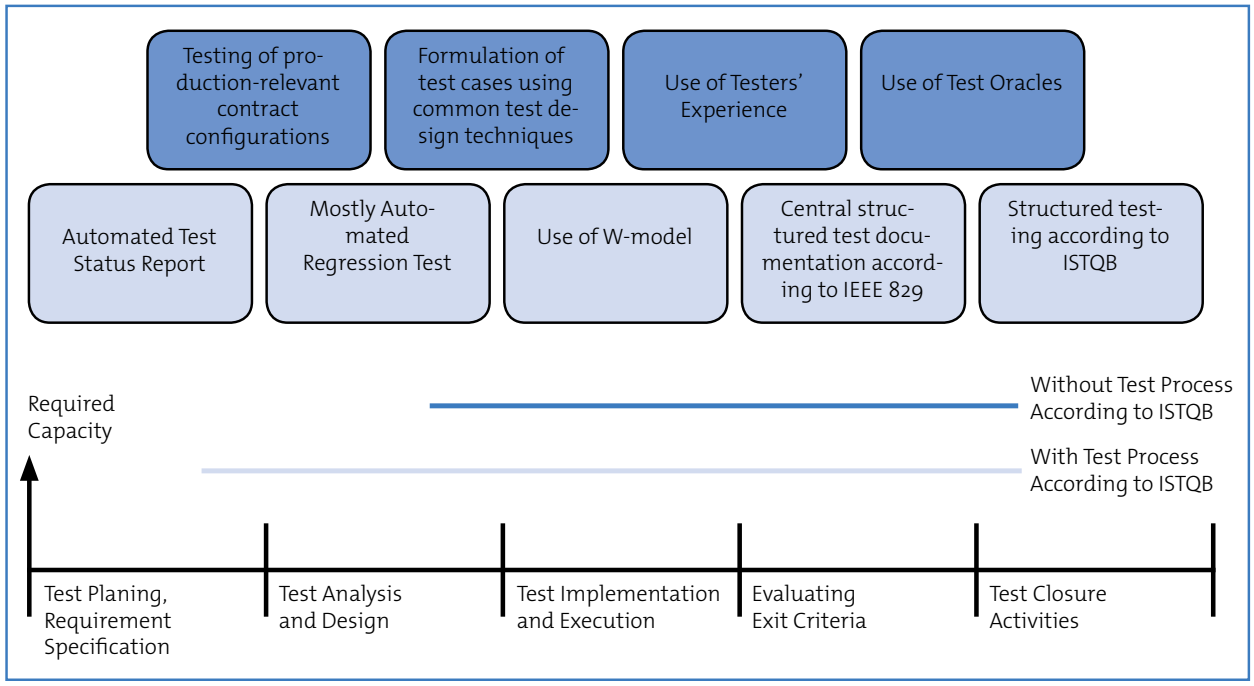


Figure 7: Test Efforts and Characteristics With and Without Test Process According to ISTQB

Conclusion

The development of test cases using common test techniques and the testing of production-relevant contract configurations are common test procedures in the insurance industry. Because of the complexity of the mathematical calculations, test oracles that can calculate the course of an insurance are also used frequently.

Unfortunately, in practice the subjects addressed above, like the structured test procedure and the use of the V-, or even better W-model, are seldom applied consistently. Often, there is no time for the test preparation tasks (analysis and design). Structured testing according to the ISTQB test process can improve this significantly. Also, test documentation is mostly not consistent and hardly suited for automatic evaluation. Consistent documentation can be achieved by test documentation according to standard IEEE 829. With it an automatic test status report can be created more easily.

The formal test procedure according to ISTQB does not require more time than the procedures used up to now. In the test process according to ISTQB, testing has to start earlier but requires less time, especially towards the end of the project. According to the W-model testers can start test design in parallel to the requirement specifications, thus ambiguities in the specifications are clarified early on. Because of this no, or at least no serious, need for clarification occurs during the actual testing. Also, when using this procedure the versions distributed by development are usually less defective. Thus fewer cases tested negative have to be corrected and tested again. Avoiding multiple runs thus positively affects the progress of testing.

During the development phase the tester has time to program the required test tools (test oracles, tools for regression testing, and the like), so that these are available at the first release of the development. Thus tests can already provide good results at the first release and be executed faster.

Figure 7 compares the efforts for testing using the formal test process according to ISTQB to those without using the formal test process.

A standardized test documentation makes it easier for testers to document their test cases. Using suitable tools the majority of the values to be documented can be automatically assigned. Thus the effort for documentation can be reduced, although more is being documented. Based on our experiences from projects, we can say that this documentation procedure guarantees auditing acceptability. In projects where we implemented the detailed and automated test documentation and test evaluation, this was praised by external and internal auditors.

Also, it must be considered that this test procedure in mathematical testing in the insurance industry requires qualified testers, who have acquired both the technical knowledge of the insurance products and the structured testing procedure.

According to our experience, the basic test process and the corresponding documentation according to the documents defined in standard IEEE 829 improves the quality in the mathematical testing of portfolio management systems significantly. Also, it is more cost-efficient, because production problems that are expensive to solve are avoided. This effect is multiplied with recurring activities, e.g. when several releases are introduced simultaneously.

Biography

Jens Fricke is a senior consultant at viadico ag. For more than 10 years he has been working as test analyst and test manager in several projects for implementation of portfolio management systems in the insurance industry. Jens received a diploma degree in mathematics (Dipl.-Math.) and the degree of an actuary DAV (German Actuarial Association). He is ISTQB Certified Tester and a member of ASQF e.V.

Thomas Niess is a consultant at viadico ag. He holds a degree in Economic Mathematics (Dipl.-Math. oec.) from the University of Ulm, with main focus on Finance and Actuarial Science. After his graduation he started his career at Finincon GmbH, Filderstadt, which in 2007 merged into viadico ag. He works as a consultant in mathematical testing of portfolio management systems in life insurance. Thomas is ISTQB Certified Tester and a member of ASQF e.V.

Interview with Chris O'Malley

by Rex Black

CA (formerly Computer Associates) has been carrying out major quality improvement initiatives since 2006. This includes a new software development methodology, PMI-based project management, testing, quality assurance, and CMMI-based process improvement. Rex Black, President of RBCS, has worked with CA to help design and execute these initiatives. Recently, Rex had a chance to talk to Chris O'Malley, Executive Vice President and General Manager, about CA's commitment to quality, how Chris sees that commitment transforming CA's software, and what the benefits will be to CA customers.

Question: So, Chris, CA is more committed than ever to quality. What do you want Testing Experience readers to know about that commitment?

Answer: We are entirely focused on ensuring the highest quality and consistency of our customer's experience. The foundation of a world-class customer experience is based on maintaining the highest levels of quality. Not only are we committed to maintaining the quality for each of our applications, but we are also committed to delivering a family of products that interoperate smoothly together. In addition, performance, security, usability, and accessibility are all key attributes for our customers, that we are committed to delivering.

Question: Chris, those sound like good commitments. How does CA intend to translate those commitments into actions?

Answer: To set the direction, we have adopted a quality assurance policy. To implement that policy, we are funding several initiatives to improve quality. These initiatives will ensure that our tactics, both in terms of day-to-day activities on projects and step-by-step long-term process improvements, are aligned with our efforts to be best in class.

We are investing significantly in testing and quality training; as you know we have the RBCS team delivering training worldwide, including the ISTQB Foundation courses, and we have RBCS e-learning courseware available enterprise-wide starting in November.

To make sure we are efficient, we are investing in testing centers of excellence in Prague, Beijing, and in Hyderabad. These are not just low-cost operations. We are providing the same level of training and skills growth to these QA Engineers to ensure the highest quality and consistency of our customer's experience.

Another efficiency dimension of our quality initiatives is our focus on test automation. We are taking new and innovative testing practices developed by some of our leading product teams, and propagating that expertise worldwide. For example, you co-wrote an article on our WATIJ effort in the Bellevue offices, and Andy Brooks and his team contributed a sidebar to your new edition of Managing the Testing Process that discussed the test management automation in place within his group.

Question: Sometimes quality initiatives like this lose momentum due to lack of senior management support. What level of executive sponsorship does this initiative have?

Answer: Our quality initiatives have the highest levels of support. Ultimately, these are sponsored by Ajei Gopal, an Executive Vice President who reports to the CEO. As you know, Rex, having met Ajei, he is very focused on quality.

Now, let me tell you why this initiative has staying power. Our quality initiative is a key element of our brand. CA wants to be known for excellence in customer experience. We believe, appreciate and understand the value that quality brings to our customer experience.

One reason for loss of momentum is the lack of measurable results. In our case, our initiatives include putting in place a comprehensive set of testing and quality metrics that will provide clear, measurable return on investment. As we move forward with this effort, we are seeing clear, objective evidence of the effectiveness, efficiency, and quality of the improvements we're delivering to CA—and to our customers.

Question: Chris, how will this change CA's software?

Answer: Well, as I mentioned, we are committed to best in class customer experience which is derived from the highest quality in terms of functionality, usability, accessibility, performance, interoperability, and security, across our entire suite of applications. We are making bold commitments to our customers and holding ourselves responsible. We are implementing a comprehensive set of testing and quality metrics to measure ourselves against these high standards.

As we measure how well we are delivering on that commitment, we continue to find opportunities to deliver even better on that commitment. We are very focused on measuring our customers experience and quickly responding to resolve their issues. The quality of CA's software will continuously improve as time goes on.

Question: Do you see this focus on quality accelerating delivery of software or slowing it down?

Answer: Our quality assurance policy is based on the idea that quality is not about testing out a huge mass of bugs at the very end of each project. That's the least effective and least efficient way to go about quality. We are building quality and assurance into the entire lifecycle. This includes projects that are following traditional lifecycles and those following Agile lifecycles. As we get better at preventing defects, or at least removing them closer in time to the moment of introduction, we expect to see project timelines shorten, because we'll be delivering fewer bugs to the test teams and beta customers at the end of our delivery cycle.

Question: In the opening years of this decade, Microsoft went through a searing experience with the poor quality of their software, particularly in terms of security, that forced them to make dramatic changes in the way they build software. Chris, do you see this focus on quality as a natural evolution in the way CA builds and sells software, or is it revolutionary?

Answer: Given that our products are developed for enterprise customers to manage their critical infrastructure, our customers have always expected a higher level of product quality and reliability. These expectations continue to grow and evolve as many of our mainframe customers are faced with a reduction in the size and experience level of their mainframe workforce. We spend a lot of time listening to our customers. They've told us what they want in terms of quality. Clearly, we must be best in class and we must build this focus into our culture. We are empowering our people to plan, manage, and execute these changes.

Question: So, let's look at this from the customer's point of view. How do you see this initiative benefiting CA's customers in term of quality? You clearly think it'll be measurable, not just subjective?

Answer: Absolutely measurable, and subjective as well. Subjective, in that people using our software in companies around the world will feel better and better about the quality of our software as this initiative continues. Measurable, not only from higher levels of customer satisfaction surveys, but we are seeing reduced field failures, fewer customer calls, greater participation in our Beta program, and ultimately, more revenue as our existing and new customers select CA products based on the overall value we can offer. That value includes quality as an intrinsic part of our brand.

Question: How are you going about picking the particular metrics?

Answer: We started by identifying the root causes that were driving customer satisfaction. We learned that product quality was a primary driver, both in terms of the number of issues and the time to resolve those issues. We found that as the Mean-Time-to-Resolution (MTTR) improves, so does customer satisfaction. As such, we are using MTTR metrics to help transform the behavior of our development organization to improve responsiveness to customer issues. Over the past two years, this focus has reduced our average MTTR by over 50%.

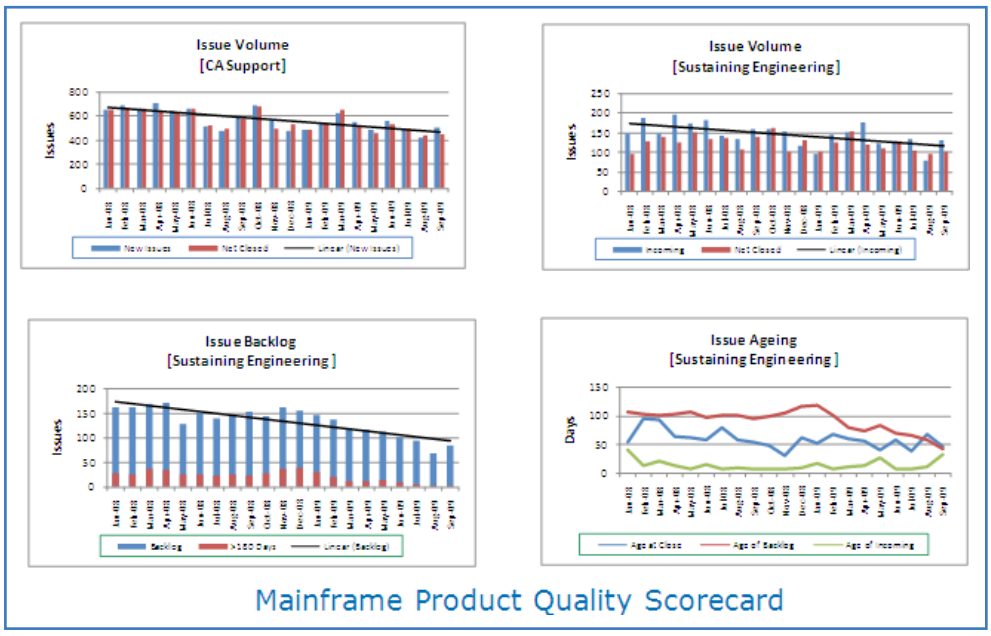
We also developed a Balanced Product Quality Scorecard that reports on key quality metrics for each of our products, based on customer support metrics. We have two levels of customer support: Level 1, for initial reporting of issues, and Level 2, for resolution of the defects underlying some of these issues. Our scorecard provides a quick view of Incoming Customer Issues to Level 1, Incoming Customer Defects to Level 2, Defect Backlog in Level 2, and Defect Aging in Level 2. Over the past 2 years, this focus has helped our mainframe products to reduce incoming issues to Level 1 by 21%, incoming defects to Level 2 by 26%, defect backlog by 57%, and defect aging by 55%.

Question: That sounds like a great tool to manage quality from the customers' perspective. Chris, do you suppose we could show readers an example of that dashboard?

Answer: Yes, we've been very pleased with how it's worked for us, and we'd be happy to share it with Testing Experience readers. [See Figure 1 for example of the dashboard.]

Question: Chris, you are responsible for mainframe products. Why do you think testing and quality is particularly important for those products?

Answer: We are seeing an aging workforce in the mainframe world. Many people who have over 30 years of experience in making these mainframes work properly are retiring. This is happening at time when mainframe usage is growing at a healthy pace. Our customers can't rely on experience, institutional knowledge, and wizardry to keep these important corporate assets going. They need us—they need CA—to make products that help them use mainframes effectively and efficiently, even with a less-experienced IT team. Our quality initiative is a key part of making sure that our mainframe customers have the tools they need as this mainframe brain drain continues over the next few decades.



We are the leading independent software vendor in the mainframe space. This brings us the responsibility to continue driving growth and innovation in mainframe applications. We need to enable the next generation of mainframe developers to scale, advance, automate and dramatically improve the customer experience of mainframe applications. We need our products to take on more of the burden from our customers due to the increasing turnover in their mainframe workforce. We need to remember that at CA, job number one is ensuring a best in class customer experience.

Applications of Model-Based Statistical Testing for the Automotive and Factory Automation Domains

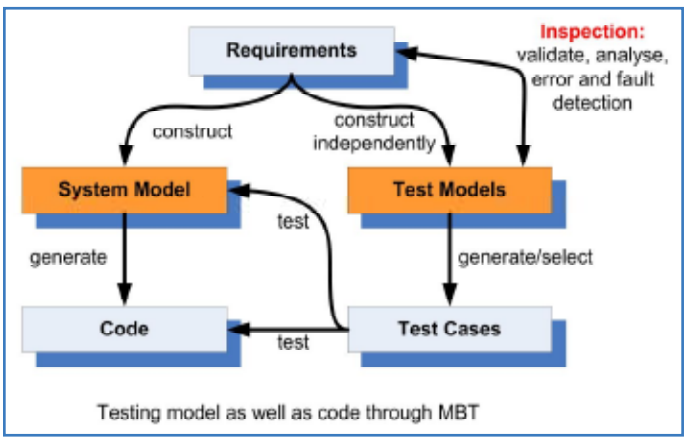
by Robert Eschbach & Tanvir Hussain

In most of the recent innovations in the automotive and automation domains their software systems played an influential role. To match with the rising proportion of software and their safety-critical role, system testing of this software is becoming more and more important. System tests are usually based upon requirement documents. Quite often, behavioral models built up through analysis of the requirement documents serve as the basis for system testing. These models are often depicted through formal or semi-formal means such as automata, Petri-nets, UML diagrams, etc. This enables a considerable degree of automation of the testing activities and provides the opportunity to quantitatively evaluate the progress of the testing process. Another important advantage of using models is to eradicate any ambiguities and incompleteness occuring in the requirements. A systematic approach of model construction as well as tool support for it appears to be immensely helpful in pinpointing the loopholes of the requirement documents. This article presents the concept of sequence-based specification to systematically construct models of software systems and illustrates its tool-supported application through certain case studies of the automotive and automation domain. The model construction procedure ensures that the resulting model is complete and traceable. The syntax and formal semantics of the model allows the automated generation of test cases quite naturally.

As far as functional system tests are concerned, the goal is to reveal errors in the software. From the perspective of the system model, it translates to achieving structural coverage of it. Since in most of the practical cases this amounts to a formidable expense, trade-offs are often made to ensure optimal, yet affordable quality assurance through testing. Statistical evaluation of the tests appears to be a useful tool for making such trade-offs. It is based upon the information regarding usage of the software and thus prioritizes rigorous testing of more frequently used functionalities. The statistical approach of evaluating test execution, applied in conjugation with the model-based testing, appeared to be a useful technique in recent years. Obviously, in safety-critical systems safety functionalities should be treated differently to ensure that they are tested exhaustively or according to the requirements of safety standards like IEC 61508, ISO 2626-2, IEC 13849.

Model-Based Statistical Testing

Model-Based Statistical Testing (MBST) refers to the enhancement of Model-Based Testing (MBT), where statistics plays an important role during selection and overall test evaluation. The term model in MBT refers to a simple, transparent and comprehensive representation of the specification. Since it is advantageous to have the model in a visual, executable and analyzable form, graphical modeling languages, such as state machines, Petri-nets or UML diagrams, are commonly used. Visual attributes of the model renders comprehensiveness and sometimes provides a better and concise way of providing information. The most desirable attribute of a model is its ability to be executed and analyzed. A model owes this attribute to the formal semantics of the underlying modeling language used. To have an executable and analyzable model means that it can serve as a prototype and it can reveal certain identified characteristics of the system. Compared to the textual specification documents, the model offers a better grasp of how the system should behave. Moreover, through trial executions one can be sure whether the model is specifying correct behavior or not .Through analysis one can find out some characteristics of the model. Finally, when one has completely built the model, it can be used to generate the test cases, each of which can be related to an execution of the model. The model which is used for test case generation is often named as test model to distinguish it from the model which is used during the design phase of Model-Driven Development (MDD). Since the test models are constructed independently, it can also be used to test the system model as well as the code generated from the model, in MDD. Throughout the years MDD came out to be a common practice for software development in the automotive domain. In the domain of industrial automation, the use of MDD is gradually gaining popularity.



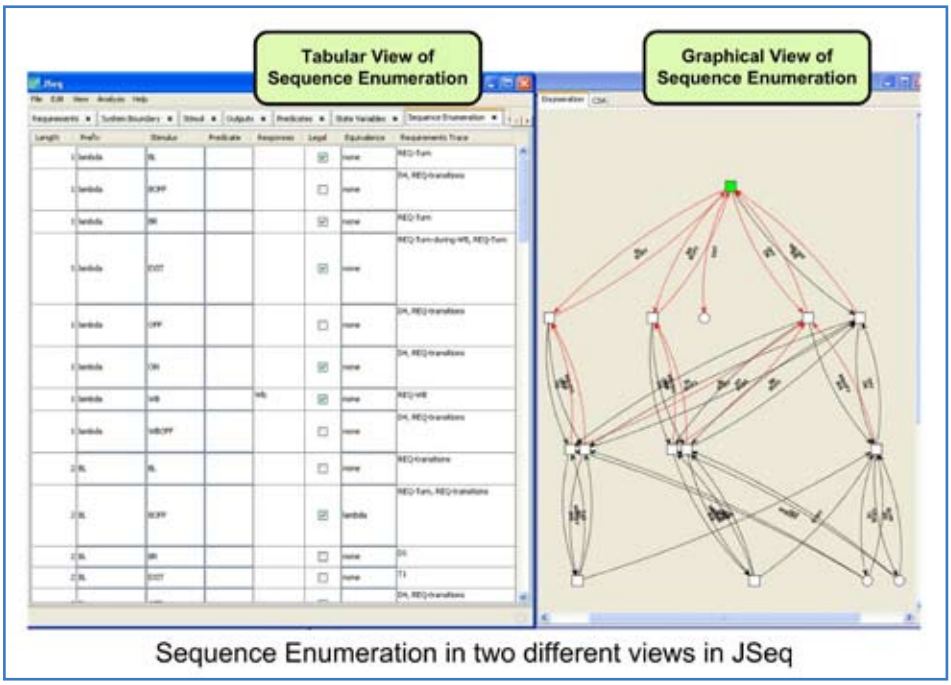
Testing model as well as code through MBT

As far as test case generation is concerned, it is certainly desirable that the model is consistent with the requirements and that each of the test cases can be related to certain aspects of the requirements. Furthermore, only requirements are captured in the test

model, such that each model property can be traced back to the requirements. One implicit purpose of constructing the models is such that due to the formal semantics of the model and due to the traceability between model and requirements, certain incompleteness or ambiguities of the textual requirement are caught and preferably eradicated. In the following, we will discuss a technique which appears to be conducive for constructing the test model in a systematic way.

Sequence-Based Specification

Sequence-Based Specification (SBS) refers to a set of techniques for stepwise construction of black-box and state-box specifications of software systems. A black-box specification is a mapping, which associates a stimulus history with its associated response. It defines the required external behavior of a system. A state-box specification is a refinement of the black-box specification that maps current stimulus and current state to response and state update. With SBS one can analyze the completeness and consistency of the requirements with a stepwise construction of a traceably correct black-box specification.



Sequence Enumeration in two different views in JSeq

To derive the SBS, one should first identify the system boundary, i.e. interfaces, stimuli and responses. Then begins the process of sequence enumeration, which aims at systematically writing down each possible stimulus sequence, starting with sequences of length one. A sequence of stimuli represents one history of the usage of the system under test. For every sequence, it is determined whether the sequence is legal or illegal and whether a corresponding response is given. The notion of an illegal sequence is attached to the one which possibly would not appear at all in the system, or which does not have any significance from the functional perspective of the system. If a sequence is marked as illegal, all sequences having this sequence as prefix are illegal as well. A legal sequence does not necessarily lead to a response.

Every sequence is compared to previously analyzed sequences with respect to equivalence. Two sequences are equivalent if their responses to future stimuli are identical. Each sequence, which can be reduced to an equivalent, previously analyzed sequence, does not need to be extended in the enumeration. The enumeration stops when there is no sequence left to expand. This happens when the newly expanded sequences are either illegal or reduced to equivalent sequences.

In the sequence enumeration, the legal and unreduced sequences are termed as canonical sequences. The canonical sequences form the states of the state-box specification. Each of the canonical sequences is further characterized by allotting unique assignments to certain variables termed as state variables. Two canonical sequences having the same assignment implies that either they ought to be equivalent, but mistakenly the equivalence was not assigned during sequence enumeration, or they need to be differentiated by defining a new variable which should assume different values for the considered sequences. Thus finally, through the analysis of the canonical sequences, one obtains a state machine, more specifically, a mealy automaton. Each of the identified canonical sequences is a state in the mealy automata, and a transition between any two states has as input the stimulus that, when suffixed to the source canonical sequence, forms either the destination canonical sequence, or another non-canonical sequence equivalent to the destination canonical sequence.

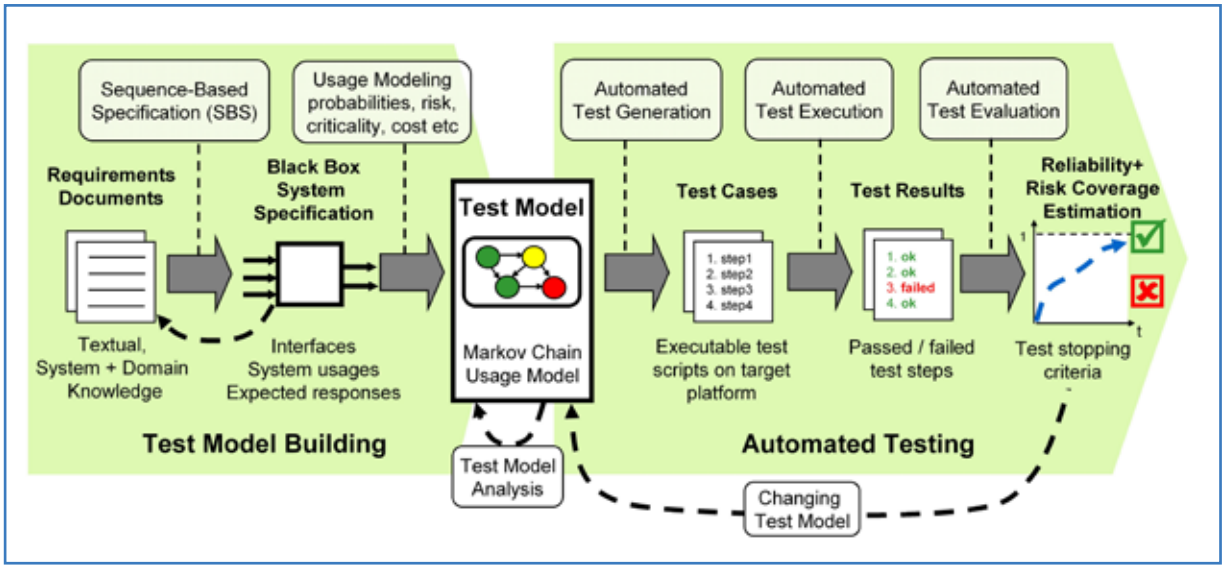
Certainly, the advantages of SBS can be enjoyed when enough tool support is available. Testing and Inspections department of Fraunhofer Institute for Experimental Software Engineering (IESE) has developed the tool JSeq for this purpose. Both the sequence enumeration and canonical sequence analysis can be performed simultaneously through the tabular editor or its graphical counterpart. Prudent and efficient algorithms are implemented in the tool to bear the consequences of the changes in any of the lists namely for stimuli, responses, state variables or of the changes of one or more properties of any sequence on the complete sequence enumeration and state-box representation.

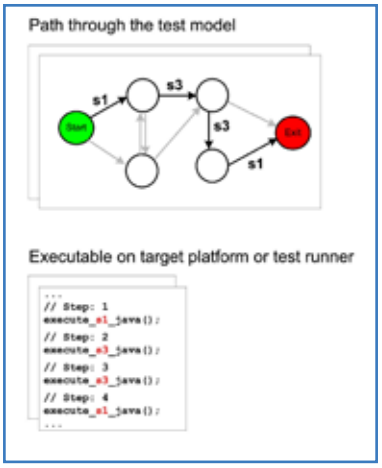
Usage Modeling

Applying SBS techniques one can construct a mealy automata test model. Further, one can annotate the transitions of the automata by probabilities in order to model the usages of the software system. A probability value allotted to a transition implies how probable it is to traverse that transition from the source state of the transition. Therefore, the probabilities of all the outgoing transitions of a state should sum to 1. Apart from probability annotations, it is also possible to assign annotations implying risk, criticality, cost etc. related to the transitions. Software systems that may undergo differing usage scenarios may have multiple probability or other annotations attached to each transition corresponding to each of the usage scenarios. With probability annotations on each of its transitions, the test model becomes a Markov chain. Analyzing the Markov chain or the test model it is possible to reveal certain characteristics which then can be validated against the requirements and in case a contradiction arises, the test model and corresponding Markov chain can be improved to eradicate the contradiction.

Automated Test Case Generation and Selection

Test cases are certain paths through the test model beginning from the initial state to the final one. That is why test case generation is quite straight-forward. To claim sufficiency or to have a certain quantitative measure of how much the subject is tested, usually tests are selected based upon certain criteria. These criteria can be based upon the structure of the model such as state or transition coverage of the test model. Based upon the annotations imposed on the test model during the usage modeling, it is also possible to define certain test selection criteria. For example, using the probability annotations one can select a definite number of test cases that are most probable. When other annotations are taken into consideration, similar selection criteria can also be defined. Moreover, tracing back the relation between the test model and the requirement document, it is also possible to select





test cases such that all the aspects or certain major aspects mentioned in the requirements document are tested at least once.

The test cases generated in MBT are mostly abstract. This is due to the fact that while constructing the test model certain abstractions are adopted for achieving simplicity and conciseness. These attributes help in understanding and analyzing the test model easily and keep the automated test generation and selection traceable. But while executing the test cases, these abstractions are to be elaborated. Most often the abstractions are related to the stimuli and responses, and it suffices to map them to tangible input and output signals. In addition to that, test oracles must also be written such that the actual responses can be compared meaningfully to the expected responses, in order to decide whether the test passed or failed. These activities are usually performed manually.

Test Evaluation

There are certainly two aspects of test evaluation in MBST. One is simply related to one or the other coverage criterion of the test model. The other one is related to the statistical aspect of the test model imposed through usage modeling. Test evaluation based on a coverage criterion gives a statement regarding the coverage of the test model or related artifacts or properties (like requirements, risks etc.) achieved through test execution. while the statistical measure presents a reliability estimation of the system under test. An example of such a statistical metric of test evaluation is single-event reliability, which describes the probability that the next randomly chosen stimulus will not invoke a failure. Another such metric is single-use reliability that represents the probability that a randomly-selected use of the software will be without failures. The tool JUMBL[1] can be used to perform such statistical test evaluations.

Case Study 1: Automotive Software

The object of this case study is the power window control unit (PCU) of a medium-sized car. Following the trend of MDD, the system model was initially developed in Simulink[2]. This served as the base for code generation, which finally got deployed on the target platform. The goal was to test the model and the deployed software, meaning to perform Model-in-the-Loop (MiL) and Hardware-in-the-Loop (HiL) tests for the system.

The test model was first prepared using the SBS technique, where a fictitious specification resembling the proprietary requirements statements in terms of complexity, described functionality, and notational format was taken as basis. The project was limited to the power window control unit (PCU) for demonstrational purposes and included manual window movement, automatic window movement, as well as jam protection. During the application of the SBS technique it turned out that 91 requirements were directly stated in the requirements document and 7 requirements were missing. Some inconsistencies, missing specifications, unclear requirements, and vague formulations were found in the requirements document during sequence enumeration. For each sequence, the requirements were analyzed with respect to response and equivalence. In some cases, the missing information could be derived from existing information, in which case it was introduced as a derived requirement. Otherwise certain plausible assumptions were made which could have been better resolved by resorting to experts.

Power window Control Unit (PCU) with HiL Test setup

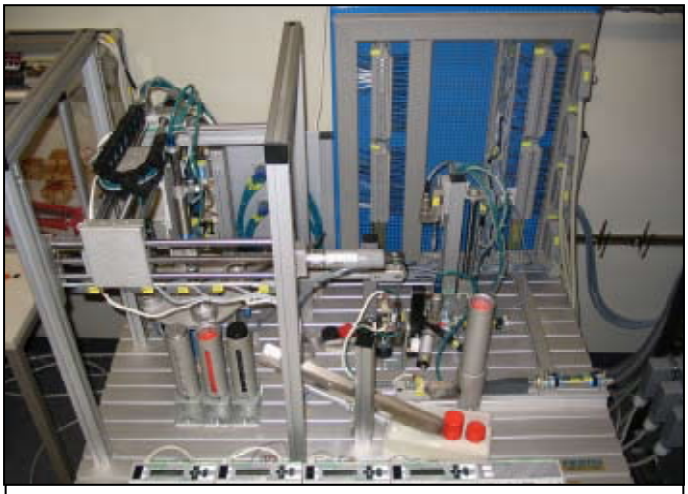
The test model obtained finally contained 9 states and 68 transitions. The test model was annotated with probabilities during usage modeling. The test suite comprised of 2 test cases for achieving the state and transition coverage of the test model and 100 randomly generated test cases consisting of 6389 test steps in total. After generating the test cases, the abstract stimuli were interpreted to continuous signals to perform the MiL tests on the

1 www.cs.utk.edu/sqrl/index.html
2 www.mathworks.com/products/simulink/

Simulink platform. To check whether the tests passed or failed, responses were checked within a defined time window. As far as the desired output signal shape came out within the time window, the test was marked as passed, otherwise those were denoted as failed. For running the HiL tests, PROVEtech TA test runner was used, which offers a VisualBasic-like test script language, called WinWrap Basic. The executable test cases contained initializing and signal-generating operations corresponding to the stimuli and consequent response evaluation within a pre-defined timing interval. Arrival of the proper response was then evaluated to a pass and violation of that as a fail. A statistical evaluation of the test executions was performed, and it turned out that the 100 random test cases were quite sufficient to achieve the desired level of confidence.

Case Study 2: Industrial Automation Software

The object of the second case study is a didactic modular production system. Over the years this system became an object of didactic and research interest due to its resemblance with real systems. In the domain of factory automation, programming languages according to IEC 61131-3 and IEC 61499 standards are predominantly used to program the controllers for such systems. The production system is divided into four modules and can operate in four different modes. The independence of operation of the modules allowed considering the control software of each module separately for testing. As the first module was considered for testing, a corresponding test model was constructed using SBS. The test model contained 23 states and 87 transitions. The test suite contained 23 test cases for covering each of the transitions and states at least once. Furthermore, random test cases were generated, and it was observed that actually 100 passed random test cases were sufficient to achieve a high enough confidence level. The test execution was performed solely on IEC 61499 compliant software. For mapping the stimuli and for evaluating the outputs, certain IEC 61499 compliant modules were developed and used.

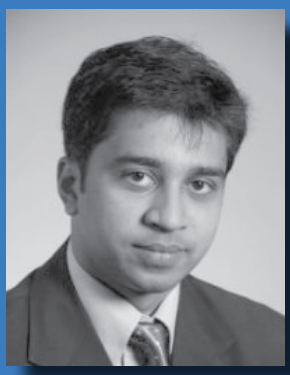


Modular Production System (MPS)

Conclusion

This article gives a short overview of MBST and presents certain successful case studies from two different domains namely automotive and industrial automation. It also provided a glimpse of the SBS technique for systematically constructing complete and consistent test models. With tool support this technique proved to be an influential element in yielding success for MBT in general. However, the greatest potential of MBST lies in delivering quantitative statements regarding the quality of the software systems, and since the statements are revealed through tests they deserve better acceptance. This certainly is an important aspect when evidence of quality of software systems is being increasingly requested, due to their safety-critical role in our daily lives.

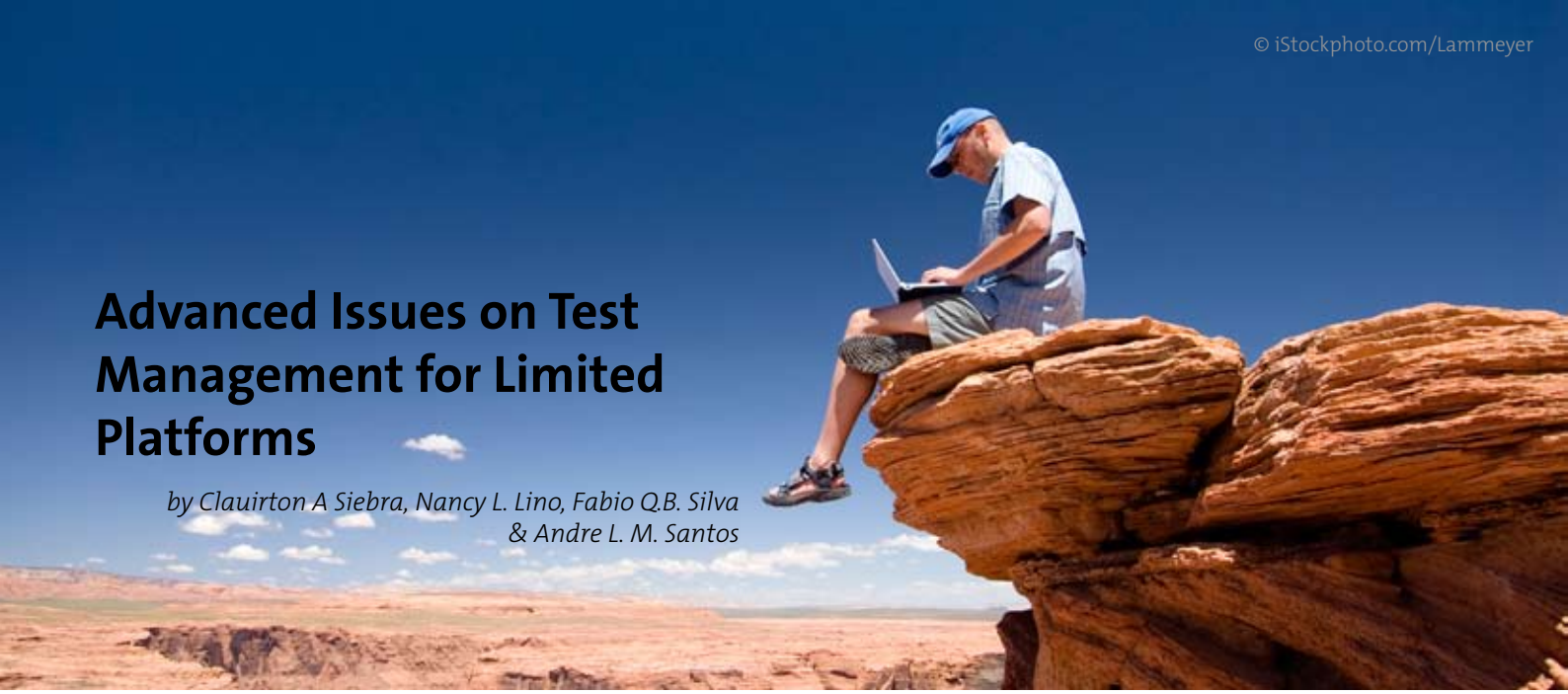
- **Foundations of Sequence-Based Software Specification**; J. S. Prowell, and J. H. Poore; *IEEE Transactions on Software Engineering*, 29(5), pp. 417–429, 2003.
- **Cleanroom Software Engineering: Technology and Process**; J. S. Prowell, C. J. Trammell, R.C. Linger and J. H. Poore; *Addison-Wesley Professional*, 1999.
- **Risk-based Testing of Automotive Systems by Applying Model-based Approaches**; R. Eschbach, J. H. Poore, T. Bauer and J. Kloos; to be presented on the 1st Commercial Vehicle Technology Symposium, Kaiserslautern, Germany, 16th-18th March, 2010.
- **Risk-based Statistical Testing: A novel approach for the reliability analysis of safety-critical embedded systems**; R. Eschbach; to be presented on the Embedded World Conference, Nürnberg, Germany, 2nd-4th March, 2010.
- **Die Zukunft in der Softwareentwicklung**; P. Liggesmeyer, R. Eschbach, R. Kalmar, M. Trapp, D. Zeckzer; *Elektronik Praxis*, 2009; available online at: http://www.elektronikpraxis.vogel.de/themen/embeddedsoftwareengineering/management/articles/239013/index.html.
- **Statistical Testing of IEC 61499 Compliant Software Components**, T. Hussain and R. Eschbach, *Proceedings of IFAC Symposium on Information Control Problems in Manufacturing (INCOM'09)*, pp.666-671, Moscow, Russia 2009.



Biography

Robert Eschbach is the testing and inspections department head at the Fraunhofer IESE, where he is mainly engaged in projects with safety-critical and software intensive systems. He is also responsible for the practical application of formal techniques in industrial projects. Dr. Eschbach earned his Ph.D. in Informatics at the University of Kaiserslautern, Germany.

Tanvir Hussain works as a senior engineer in the testing and inspections department at Fraunhofer IESE. He earned his Ph.D. in Electrical and Computer Engineering at the University of Kaiserslautern, Germany.



Advanced Issues on Test Management for Limited Platforms

by Clauirton A Siebra, Nancy L. Lino, Fabio Q.B. Silva & Andre L. M. Santos

During recent years, the wireless network has presented a significant evolution in its technology. While first-generation networks were targeted primarily at voice and data communications occurring at low data rates, we have recently seen the evolution of second- and third-generation wireless systems that incorporate the features provided by broadband [3]. In addition to supporting mobility, broadband also aims to support multimedia traffic with quality of service (QoS) assurance so that the evolution from 2G to 3G wireless technology bears the promise of offering a wide range of new multimedia services to mobile subscribers [8]. In this context, handset devices are following this ongoing network evolution and taking advantage of this technological update to offer a broad variety of resources and applications to their users. In fact, the handset software development has evolved into a complex engineering process, mainly because of the recent network capabilities supported by new mobile communication and computing advances (memory, processing power, better resources for information delivery, etc.). This scenario has increased the pressure on the test stage of the handsets development, which is required to apply more extensive and efficient procedures of evaluation so that the final product meets the fast time-to-market goals and that it can compete in the global marketplace.

While number and complexity of tests are increasing, test centers need to improve their test process time. Note that the faster a specific handset is evaluated and delivered to the market, the better will be its chances against other models. Thus, we have a contradiction: we need to increase the number of tests and decrease the test time. Furthermore, this contradiction can lead us to reduce the quality of our test process.

The use of test management solutions, which are able to support several stages of a test process [7], is an option to ensure a better control and quality of this process. In fact, there are several options for management tools available in the market [1]. However, it is hard to cover all the stages of the test process with a unique tool, particularly if the test domain differs from the traditional software development cycle. Considering this fact, we have investigated and specified a test solution, according to the requirements of our limited platform domain. This solution was carried out in a modular way, so that each module could be instantiated with third-party or home-made solutions.

Abstraction of the Test Architecture

Test management architectures can be seen as a set of several different modules. Each of them is a computational process that intends to perform a function related to the whole test process. The diagram in Figure 1 shows an abstract view of test management modules that we consider important to our test process. This architecture stresses six main test modules (summarized below), together with its iteration with other components (database and reports):

- TC Generation: accounts for populating the TC Database with test cases that validate the *Domain Specification* [9]. This module can be an automatic process if the domain specification is modelled in a formal way. Approaches in this direction are mainly based on formal methods [5];
- Mapping: accounts for the generation of scripts to be executed in a production environment. In contrast to test cases, which do not usually change, the script language depends on the environment in which it is going to be executed. The automatic generation of scripts can be carried out using similar techniques to those used for TC generation;
- Filter: selects the test cases that are going to be used in a test cycle, according to some *Selection Criteria*. For example, a product may not support a certain function, so that all the tests related to this function must be eliminated from the active test suite;
- Planning: creates an optimal sequence of tests (plan of tests) based on parameters such as Plan Criteria (time and resources), Priorities (simple indications of test ordering) and Historical Data rules (e.g., indications of more problematic tests to be carried out first);
- Execution: accounts for the real performance of a pre-defined sequence of tests. For this, the module sends the planned test suite to the production environment and monitors the execution of this sequence via control information. Control information could be, for example, an indication that a TC

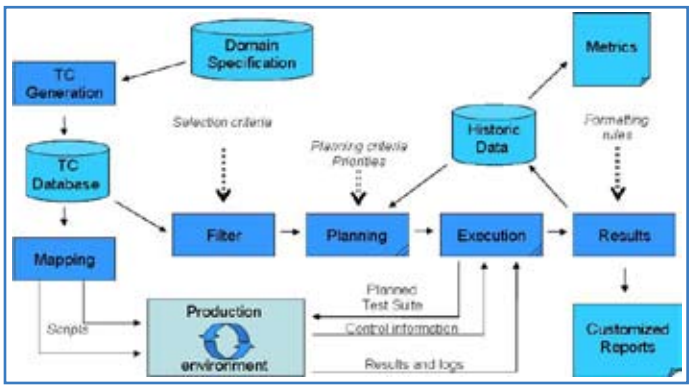


Figure 1. Test Management Architecture

has failed;

- Results: generates a customized report according to *Formatting Rules*. Such rules can be seen as templates, which are instantiated with test result data. This module also generates historic data about the test cycle. These data are important to raise metrics about the process and to lead future plan definitions. Metrics indicate, for example, average time to perform test suites, so that we have a good basis for estimating future cycles and predicting possible problems.

The *Planning* and *Execution* modules have a more complex structure (Figures 2). To create an execution sequence of tests, the planning module must act as a schedule, where a restriction manager generates constraints to be respected by this schedule. A priority modifier uses the historic data to set new priorities that can optimize the process. The execution module (Figure 2) has a set of control rules that lead the decision process in case of failures. This module also acts in situations where we could change the test sequence to optimize the process. For example, consider that some of the tests must be repeated several times and there is an associated approval percentage. For instance, each test is represented by the 3-tuple <t,η,φ>, where t is the test identifier, η is the number of test repetitions for each device, and φ is the approval percentage. Then a 3-tuple specified as $<t_1,12,75\%>$ means that t_1 must be performed twelve times, and the device will only be approved if the result is correct at least nine times. However, if the first nine tests are correct, then the other three runs do not need to be executed, avoiding waste of time.

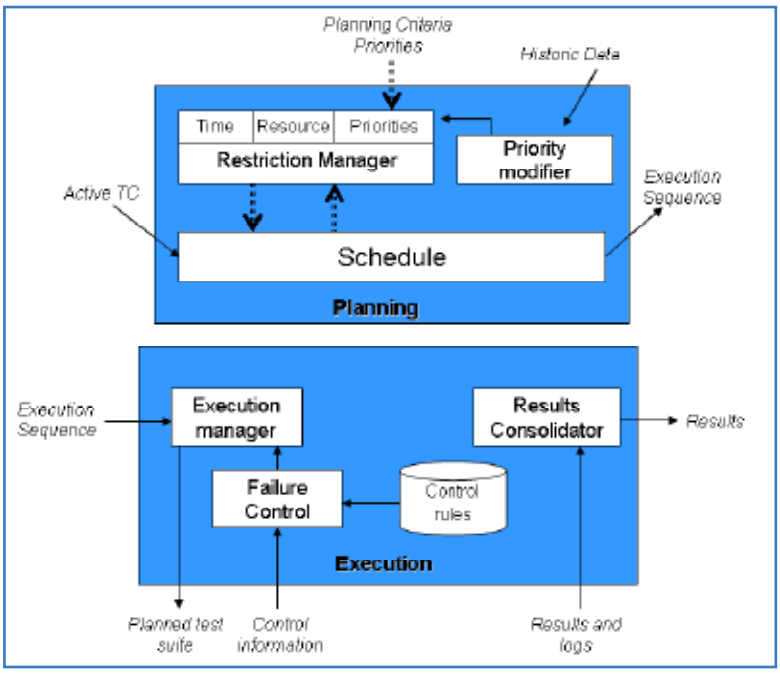


Figure 2. Details of the Planning and Execution Module

Setting the Architecture

The principal aim of this implementation was to increase the level of test automation. In this respect, we have worked with the modules of filter, planning, execution, results and production environment. Some of the modules (TC Generation and Mapping) are not considered in this paper. However some approaches for these modules, can be seen in [5].

The Role of Testlink

The Testlink tool is the backbone of our solution. Its first function is to act as the editor and organizer of test cases, saving all the related information in its database, which represents our *TC Database* (Figure 1). As the test cases were implemented in a structured way, we could apply a parser to extract the test information from the documents and insert such information into the database tables. This sort of parser is very important, because we had more than 1000 test cases to be inserted into the database. Thus, the time required to implement this parser is justified if you consider the manual work needed to populate the tables. Testlink provides an API that enables the manipulation of data via typical database operations such as insert, delete and update.

Testlink also supports the *Filter Module* functions, because it can select test cases to compose test suites according to pre-defined keywords associated with each test case during its creation. We have defined a very simple input form, where users can check the features related to a specific handset model in test. Then, using this information, we can define a SQL select query in runtime so that only the valid tests are considered in the plan test (this query verifies the keywords attribute from each test case). For example, consider a test case with the following features: 900 MHz, Voice Call and MMS. If a handset operates in the 850 MHz or 1900 MHz bands, then this testcase should be eliminated from the final test suite. Note that we occasionally need to create a *negative test set*, a set of tests that are not supported by a handset, and apply this set to observe the handset behavior during operations that are not supported by it. This set must be considered separately, and it can also be created via SQL select queries.

The third Testlink function is to support the *Results Module functions*. To this end, Testlink saves all the results information of past and current execution in a database that represents the Historic Database in our architecture. This enables the creation of several types of report related to the test execution itself and to the statistical metrics generation. In fact, Testlink already brings predefined templates, which consolidate the historic test information contained in its database. Another resource is the query metrics report frame, where we are able to perform some simple queries on the test data results, which are maintained in the database.

We have generated some reports using Testlink and observed that its reports are limited. For example, its query metrics report frame does not enable complex queries using logic operators (and, or, not, etc.). Thus, we are currently investigating some report generator tools. Some examples are *Jasper Report* and *Eclipse Birt* tools. Our initial analysis shows that both tools offer an appropriate level of flexibility, and they are an interesting alternative if more complex reports are required. Furthermore, they are also open- source projects under the GNU General Public License. Testlink also supports the activities of planning test sequences and test execution. However, this support is limited if we consider our premise of automation. The selection of tests to compose a test suite (a plan) is manually performed by testers, and they must manage details such as sequence correctness, constraints of time and opportunities for optimization. Regarding the execution, Testlink is only an input interface where testers utilize the test results to fit the interface fields. Thus, both modules should be extended to support the premise of automation and intelligent control.

Expanding the Planning Module

Testlink considers the concept of a test plan as a table in its database, so that each plan is an entry in this table. Test plans are then loaded by the execution interface so that testers can choose one and execute it. Considering the idea of automation, test plans could be built via an external component and saved in the Testlink database. To implement this idea, we have specified the *Planning Module* as an Intelligent Planning system [4], which implements a schedule of test cases as a constraint satisfaction problem (CSP). In this case, time, resources and priorities are constraints that must be respected during the development of a test plan. The interaction between Testlink and the planning module is performed via the Testlink Java API client. Using this API, the planning module can access valid test cases from the Testlink database and save test plans to the database. At the moment, the

Priority Modifier (Figure 2) changes the priority of test execution according to the frequency of errors of each test case. This information is acquired via queries in the database, since the results of all tests are saved in its tables.

Expanding the Execution Module

We are proving a level of intelligence to the *Execution Module* via the use of a cognitive function. To this end, we have specified a knowledge base and a reasoning process using JEOPS (*Java Embedded Object Production System*) [2], a Java API that adds forward chaining, first-order production rules to Java through a set of classes designed to provide this language with some kind of declarative programming. The knowledge base is able to keep an internal representation of test engineers' expertise and use such knowledge to take decisions and make choices during the test process. Thus, we can implement autonomic actions in case of failure, or as a way to improve the process when some optimization opportunity is detected. The creation of a knowledge base requires that relevant data and information can be translated into knowledge. Knowledge Engineering [6] is an artificial intelligence technique that addresses such problems. This technique makes use of some formal representation, such as rules in First Order Logic. In this sense, "real" knowledge of the world needs to be syntactically and semantically mapped into a series of conventions that makes it possible to describe things and then store them on a base of knowledge. Put in sequence, the knowledge engineer specifies what is true and the inference procedure figures out how to turn the facts into a solution to the problem.

Conclusion and Directions

In this article, we have described our experience in specifying a test management solution, which was based on the open-source Testlink tool. Our focus was on extending this tool with capabilities of automation, intelligent control and use of statistic metrics. To this end, we have specified a modular test architecture and performed some experiments using a subset of our total test suite. The advantages of using a test management solution can be observed if we analyse some process qualification parameters. First, we could maintain the same requirements coverage using a test suite that is smaller than the original one. This was observed because Testlink enables the coverage and tracing of requirements, so that it stresses test cases that perform evaluations of the same parts of the software. This redundancy is present, because some test cases require the execution of some operations that were already evaluated. Our challenge now is to use this information also as a kind of constraint in the planning process. The idea is to optimize the coverage and avoid as much redundancy as possible.

The second advantage is the support provided to the creation and maintenance of several specialized test suites, which can be applied into specific scenarios depending on the requirements of the development team. We have observed that this creation directly affects the efficiency of the planning module. If test cases are self-contained (they perform their own pre-configuration and necessary operations), then we will have a large percentage of redundancy. In this case, the planner is not able to find a test plan with a high number-tests/redundancy rate, considering a fixed total time. Differently, dependent and granular test cases are more appropriate to be used by planners, which are able to reach higher values to the number-tests/redundancy rate. Unfortunately, such test cases may require the performance of other test cases that are not part of the test scenario (test cases are sorted out by the filter module in accordance with the current test scenario). Thus, there is no guarantee that a complete test plan will be found.

Third, the quality of final reports is ensured by pre-defined templates. We can also create new kinds of templates to relate test parameters. The quality of such templates can be improved via the use of external report generation tools. We are still analyzing this alternative; however, the integration of such tools to our architecture seems to be simple, because they just need to access the Testlink database. The disadvantage is that we will have one more component rather than an integrated solution. Furthermore, using the own Testlink, all the generated reports could be accessed in real-time via Web. Note that Testlink is a web-based application developed in PHP. This fact is very important for us, because the teams involved in the development process are geographically distributed (Recife/Brazil, São Paulo/Brazil and Gumi/Korea).

Fourth, the solution improves the efficiency of the process, mainly in terms of execution time, due to the level of automation provided by its modules. For example, automation has avoided several common errors related to human manipulation. In fact, tests related to network handset evaluation are very repetitive and stressing due to the amount of required keyboard inputs, navigation and configurations. Finally, the maintenance of historic data is very important for the measurement and analysis of the quality of our test process.

In our next steps, we intend to integrate this test management solution into an autonomous execution environment, so that a percentage of 55% of the tests can be executed without human intervention. The main problem is to codify all the expertise of test engineers in facts and rules to compose the knowledge base. Furthermore, a significant number of tests requires manual execution, mainly because they need some kind of mechanical interaction (e.g., hard reset, press-and-hold operations, etc.), which cannot be carried out via software activation. A final remark concerns the interface between the production environment and execution module. This interface enables the exchange of information about planned test suite, control messages and result data. Note that this protocol must be standardized, otherwise new production environments will find problems when being integrated into the architecture. An option is to use or define a test ontology that covers all required test information. This study is an important research direction of this work, mainly because it will enable the use of this architecture in different software domains.

References

[1] Chin, L., Worth, D. and Greenough, C. 2007. A Survey of Software Testing Tools for Computational Science, RAL Technical Reports, RAL-TR-2007-010.

[2] Filho, C. and Ramalho, G. 2000. JEOPS - The Java Embedded Object Production System", Lecture Notes In Computer Science, Vol. 1952, pp. 53 - 62, Springer-Verlag, London, UK.

[3] Garg, V. 2001. Wireless Network Evolution 2G to 3G". Prentice Hall.

[4] Ghallab, G., Nau, D. and Traverso, P. 2004. Automated Planning: theory and practice, Morgan Kaufmann Publishers.

[5] Prasanna, M., Sivanandam, S., Venkatesan, R. and Sundarrajan, R. 2005. A Survey on Automatic Test Case Generation, Academic Open Internet Journal, 15.

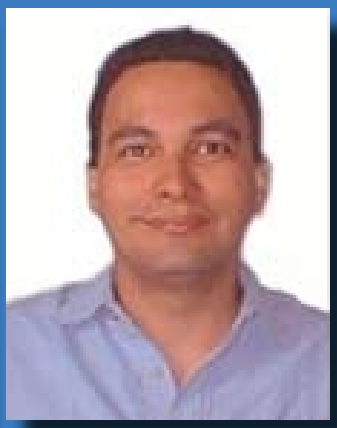
[6] Schreiber, G., Akkermans, H., Anjewierden, A., Hoog, R., Shadbolt, N., Velde, W. and Wielinga, B. 1999. Knowledge Engineering and Management: The CommonKADS Methodology. The MIT Press.

[7] Singh, H. and Bawa, H. 1995. Management of Effective Verification and Validation, IEEE Engineering Management Conference, pp.204-207.

[8] Vriendt, J., Laine, P., Lerouge, C. and Xiaofeng, X. 2002. Mobile network evolution: a revolution on the move, IEEE Communications Magazine, 40(4):104-111.

[9] Yamaura, T. 1998. How to design practical test cases, IEEE Software, 15(6):30-36.






Biography

Clauirton Siebra completed his PhD in Artificial Intelligence at the University of Edinburgh (UK) in 2006. Since then he has been working in the areas of test automation and intelligent control, as consultant of the CIn/Samsung Laboratory of Research and Development, focusing his investigations on new concepts for autonomic and test management tools. He is also a Researcher/Lecturer at the Federal University of Paraiba, leading projects related to Multi-Agent Systems and Health Informatics.

Nancy Lino has been working as a researcher in Artificial Intelligence since 2005, when she was an undergraduate student at Federal University of Pernambuco. In 2009 she received the master degree at the same university, carrying out researches involving tests and autonomic computing. At the moment, she is a software engineer at the CIn/Samsung Laboratory of Research and Development and her interests are in the field of test management tools and automation using IA techniques.

André Santos has a PhD degree from the University of Glasgow, Scotland, and is currently a lecturer at the Informatics Center of the Federal University of Pernambuco, in Brazil. His main areas of research are programming languages and software engineering.

Fabio Q. B. da Silva has a PhD in Computer Science from the University of Edinburgh, Scotland, since 1992. He joined the Center of Informatics of the Federal University of Brazil in 1993, where he is an Adjunct Professor. His areas of research and teaching are experimental software engineering and management of innovation. He has supervised over 20 master students and 2 doctoral researches.

Test Language

Introduction to Keyword-Driven Testing

by Ayal Zylberman & Aviram Shotten

KDT is the next generation test automation approach that separates the task of automated test case implementation from the automation infrastructure.

Test Language is not a test automation approach. Test Language is a comprehensive test approach that hands over the responsibility of automation plan, design and execution to the functional testers by using a KDT-based solution.

Background

Several years ago, I was Test Manager for a large organization in the defense industry. The team consisted of 10 test engineers, 2 of which were test automation experts, the others were functional testers. The test automation experts were focused on developing test automation scripts that covered some of the functional processes of the application.

It took me only a few days to realize that things were not efficient.

The same tests that were executed automatically were also tested manually. The functional testers had low confidence in test automation scripts, since the test automation experts did not have the required knowledge about the application under test. The test automation development could start only after the application was ready for testing, and thus was relevant only for a small part of the tests (mainly regression tests).

I presented the problem to the team. After many discussions, we came up with the idea of letting the functional testers create test automation scripts.

The idea may be simple, but we faced a number of challenges in trying to implement it. The main obstacle was that functional testers did not have programming skills or test automation knowledge. To solve this, the test automaters suggested that the functional testers would create their test cases using pre-defined words, which they will "translate" into automated scripts. This approach was favorably accepted by the entire team.

We proceeded to create a keyword dictionary. The keywords were mutually defined by the automation experts and functional testers. During the first phase, the functional testers wrote their test cases using the keywords, and the automation experts translated them into automation scripts. At a later stage, we developed an automated application that took care of the translation. This enabled the functional testers to create and execute the automated tests and saved even more time.

The keyword dictionary combined with the automated translation application generated a fast return on investment. This allowed us to position test automation as a basis of our testing strategy rather than a luxury.

This was the beginning of Test Language.

Test Automation

Let's review the way Test Automation is commonly approached.

The testing process includes a number of phases: test planning, test definition, bug tracking and test closure activities. The actual test execution phase is often recursive, repetitive and manual. This phase is usually described as a tedious and boring and can be carried out automatically.

To automate the execution phase, you need to create test automation scripts that are usually implemented in commercial test automation tools, such as HP's QTP, IBM's Robot, AutomatedQA's Test Complete, MS VS 2010 team system and freeware tools such as Selenium etc..

Each test automation script represents a test case or complementary steps for manual test cases. The scripts are created by test automation experts after the completion of the manual test design.

Contrary what test tool vendors would have you believe, test automation is expensive. It does not replace the need for manual testing or enable you to "down-size" your testing department.

Automated testing is an addition to your testing process. According to Cem Kaner, "Improving the Maintainability of Automated Test Suites", it can take between 3 to 10 times longer to develop, verify, and document an automated test case than to create and execute a manual test case. This is especially true if you elect to use the "record/playback" feature (contained in most test tools, as your primary automated testing methodology).

Sanity check: Measure the success of your Testing Automation project.

Step 1: Measure how many man hours were invested in test automation in the past 6 months (include Test Automation experts, functional testers, management overhead etc.). Mark this as **A**.
Step 2: Measure how many working hours would have needed in order to manually execute all tests that were executed automatically in the past 6 month. Mark this as **B**.

If A > B than most likely your Test Automation project is a failure candidate.

Why Do Test Automation Projects Fail?

These are the common reasons for test automation failures:

1. **Process** – Test Automation requires changes in the test process. The changes apply to the
   a. Test design approaches - Test Automation requires more specific design.
   b. Test coverage - Test Automation allows more scenarios to be tested on a specific function.
   c. Test execution - functions in the application that are tested automatically shouldn't be tested manually.

   Many organizations fail to plan or implement the required changes.
2. **Maintenance** – Automated tests require both functional maintenance (i.e. updating the scripts after a functional change in the AUT) and technical maintenance (i.e. AUT UI changed from MFC to C#).
3. **Expertise** – In order to write an efficient automated test, test automation experts are required to be a techno-geek version of superman – good test engineers, system experts and great SW developers.

Obviously, a different approach is required to make test automation work.

What is Test Language?

Test Language is a dictionary of keywords that helps testers to communicate with one another and with other subject-matter experts. The keywords replace the common English as the basis and create an approach called keyword-driven testing (KDT).

KDT can be used to achieve a number of goals:

- **Improve communication** between testers
- **Avoid inconsistency** in test documents
- Serve as the infrastructure for **Test Automation** based on Keyword-Driven Testing.

Test Language structure

The Test Language is based on a dictionary, which is comprised of words (keywords) and parameters.

Test Cases

A test case is a sequence of steps that tests the behavior of a given functionality/ feature in an application. Unlike traditional test approaches, Test Language uses pre-defined keywords to describe the steps and expected results (see example in Figure 2 below).

The Keywords

Keywords are the basic functional sub-procedures for the test cases of the application under test. A test case is comprised of at least one keyword.

Types of Keywords

Item Operation (Item) – an action that performs a specific operation on a given GUI component. For example: set value "Larry Swartz" in "customer name" control, verify that the value "3" appears in the "result" field. When performing an operation on a GUI item, the following parameters should be specified: Name of GUI item, what operation to perform, and the values.

Utility Functions (Function) – a script that executes a certain functional operation that is hard/non-effective to implement as a sequence. For example: wait X seconds, retrieve data from DB etc.

Sequence – a set of keywords that produces a business process, such as "create customer". We recommend collecting frequently used functional processes such as login, addition of new records to the system as sequences, instead of implementing them as items in test cases.

Parameters

In most cases, parameters should be defined for the created keywords. The parameters are additional information required in order to generate the test conditions. For example: failed authentication by-passing username with illegal password, number for mathematical calculation, etc.

Examples for sequence parameters:

create_customer (FirstName, LastName, BirthDate, email)
Keyword Parameters

When the user wants to create a new customer, the following syntax is used:

create_custmer (Bob,Dylan,1/1/2000,bobdylan@gmail.com)

Using default parameters

Some of the keywords may contain dozens of parameters. In order to simplify the test creation, all parameters should contain default values. The tester should be able to change each one of the default parameters according to the context of the test. For example, if the tester would like to create a new customer that is older than 100 years, only the birth date will be changed and all other parameters will remain the same. Obviously, a specific change will not affect the default parameters being used for other tests.

It is extremely important to plan the keywords well and to optimize the amount of keywords by creating multi-function keywords (for example, creating "Change_customer_status" keyword is a better approach than creating 2 special keywords for "activate_customer" and "deactivate_customer")

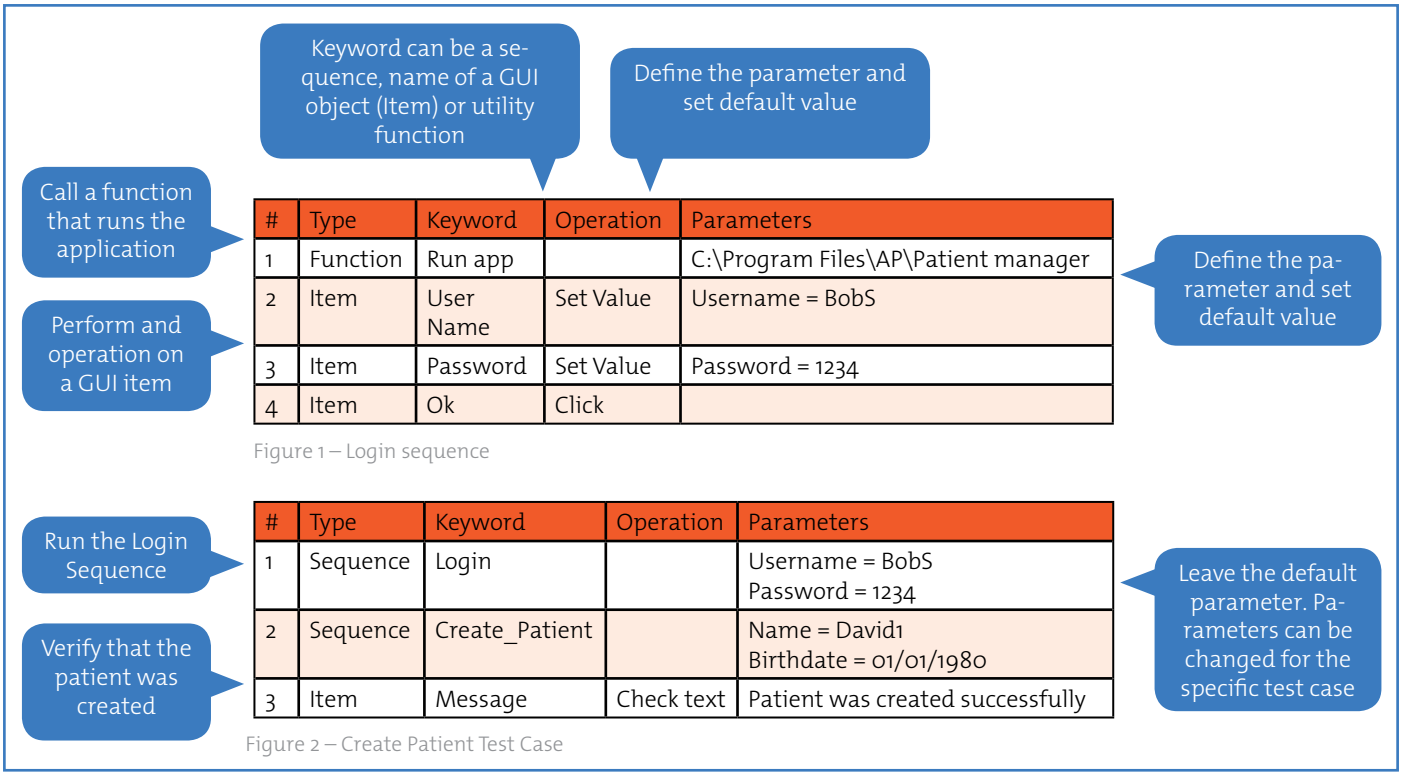
Keyword-Driven Testing (KDT)

KDT is the mechanism used in order to implement Test Language.

Keyword-Driven Testing can be divided into two main layers:

Infrastructure Layer (KDT Engine), a combination of Item Operation, Utility Functions and User-Defined Functions (also called Sequence), used as an "engine" that receives inputs (keywords) and performs operations on the application under test.

Logical Layer (KDT Test Case) - used by manual testers and other subject matter experts to build and execute test scripts by using a pre-defined keyword.



#	Type	Keyword	Operation	Parameters
1	Function	Run app		C:\Program Files\AP\Patient manager
2	Item	User Name	Set Value	Username = BobS
3	Item	Password	Set Value	Password = 1234
4	Item	Ok	Click	

Figure 1 – Login sequence

#	Type	Keyword	Operation	Parameters
1	Sequence	Login		Username = BobS Password = 1234
2	Sequence	Create_Patient		Name = David1 Birthdate = 01/01/1980
3	Item	Message	Check text	Patient was created successfully

Figure 2 – Create Patient Test Case

KDT Principles

Simplicity of test case implementation and reviewing

Functional testers should create the automated test cases directly in a simple, non-code, elite language. This eliminates the need for the time-consuming two-step process, in which testers create manual test cases which are then converted into automated scripts by the test automation team.

Automation infrastructure and test cases coexistence

The test automation infrastructure should be separated from the logical layer (which is the test cases). This allows testers to create automated test cases at an early stage in the development lifecycle before the application is ready and even before the automation infrastructure is developed.

Plan and control

In order to maximize the benefit from KDT, a plan used to assess the on-going effectiveness of the automation program.

What to Automate

Test Language should focus on testing new features rather than on regression testing, which is the common practice in traditional test automation

In traditional test automation implementations, automation scripts cannot be created before the application is ready for testing. As a result, sanity and regression tests are the main areas, which are automated. This approach usually leads to parallel executions (both manually and automatically) of the same tests.

In addition, tests, written for manual execution, are phrased in a way that does not take into account the strengths of automation scripting. The main gap between the two approaches are:

- Detail level: Test Automation requires more details (for example, after log-in, the automated script should define what is expected to happen, while in many manual scripts, this is left to the tester's intuition).
- Coverage: When performing test automation, more test cases can be created. For example: when testing a certain numeric field that get values in the range of 1-10, usually boundary analysis will be used to test the following numbers: 1, 2 , 10 and 11, while for automated tests, more scenarios can be planned.

KDT allows functional testers to plan test automation before the application is ready. This enables organizations to broaden the automated testing spectrum.

We recommend that new tests rather than translation of existing tests should be the focus of automated testing using KDT. All new tests can be planned using the KDT approach. We believe that this approach eases the KDT implementation process and greatly improves the ROI of automated testing.

Organization Structure and Responsibilities

The organizational structure of a KDT-based organization is similar to traditional test automation based organizations. It consists of Core Test Automation teams and Functional Testers. However, the responsibilities are much different:

Activity	Automation Experts	Functional Testers	Notes
Define Keywords		+	Keywords should be defined by functional testers under the supervision and control of the test automation experts
Develop Utility Functions	+		
Develop Sequences (user- defined functions)	+	+	Sequences are developed by functional testers in small-scale organizations and for private use (functions that will not be used by other testers).
Develop Test Cases		+	
Execute Test Cases		+	

Figure 3: KDT Responsibilities Matrix

KDT Process

The key factor to fully enjoy the benefit of the Keyword-Driven approach is by fully integrating it throughout the entire test process.

The following diagram shows the KDT process:

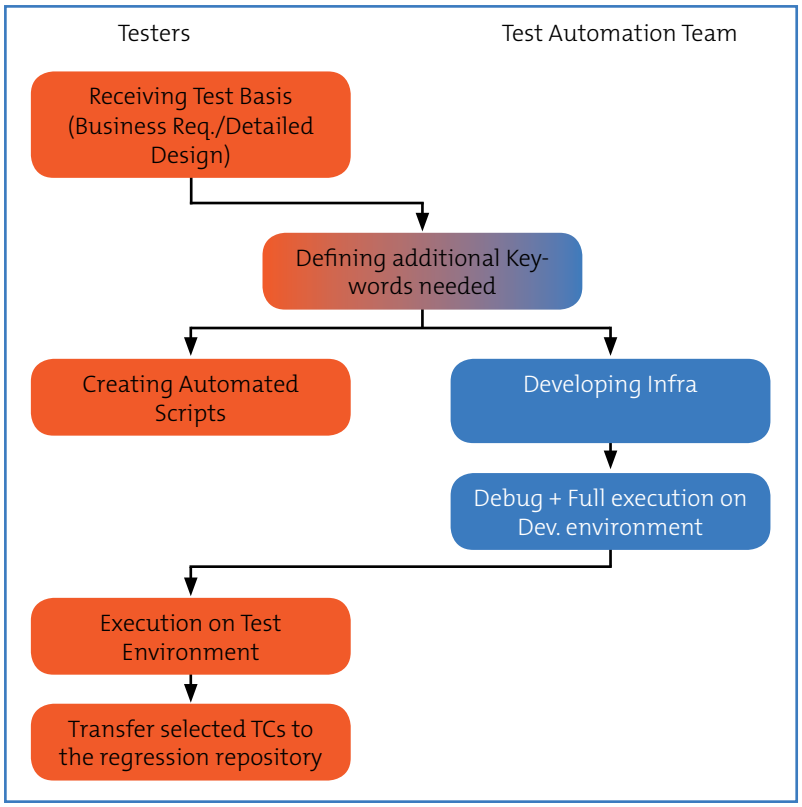


Figure 4 – Keyword Driven Testing Process

Biography

Ayal Zylberman is a senior Software Test Specialist and QualiTest Group co-founder. He has been in the field of Software Testing since 1995 testing several disciplines such as military systems, billing systems, SAP, RT, NMS, networking and telephony. He was one of the first in Israel to implement automated testing tools and is recognized as one of the top-level experts in this field worldwide. Ayal was involved in Test Automation activities in more than 50 companies. During his career, he published more than 10 professional articles worldwide and is a frequently requested lecturer at Israeli and international conferences.

Aviram Shotten is QualiTest's program manager for the "Automation Planner®" suite supporting KDT, and a senior test manager responsible for over 80 test engineers in over 25 projects; among them more than 10 have successfully implemented KDT automation via the Automation Planner®.

Test design for stubborn applications

Event handling in automated acceptance tests

by Alexandra Imrie & Markus Tiede

At the beginning of any test automation project for acceptance tests, the focus is usually on creating a set of running tests to cover the most critical or newest aspects of the software being developed. Such acceptance tests see the application as a black-box, testing the business logic through the GUI. Once a good base of runnable tests has been produced, however, it quickly becomes critically important to ensure that events that affect one part of the test do not lead to all other tests subsequently failing (due to "inherited" problems from the first failure), or not running at all (because the whole test has been aborted) . The discovery of errors in an application is a natural and desired effect of automated acceptance tests; however, the other aim of acceptance testing should always be to have the most comprehensive knowledge of the software quality. If 90% of the tests cannot run because an event occurred in the first 10%, this aim cannot be achieved. The quality will suffer as a result, and the costs to fix any subsequent untested errors will increase as more time passes between introducing the error and finding and resolving it.

A well thought-out system of event handling is therefore necessary to ensure that the quality of the whole software project can be monitored despite problems in individual areas. In addition, adding event handling to the project can make it easier to find and reproduce errors to allow them to be fixed in a short time. This article offers strategies and suggestions on introducing robust and understandable event handling in automated acceptance tests.

Definitions

The terms *error* and *exception* have specific meanings for software developers: *exceptions* can be handled and errors cannot be handled. From the perspective of the tester, this meaning holds true. Therefore, for the purposes of this article, we use the following definitions:

Event: Either an error or an exception; something that causes the test flow to deviate from the expected execution.

Exception: An exception results from a failed assertion in the test execution (for example, an expected value or status does not correspond to the actual value or status). With exceptions, the test can continue after some kind of "intervention" by an event handler.

Error: An error is an event that can't be dealt with in the test execution. Any following steps that are dependent on the success of this step cannot be executed.

It should be noted that what is an *error* for a developer (something that the software cannot handle) may manifest itself as an *exception* for the tester (something that the test execution can deal with), and vice-versa.

What types of event can affect an automated test?

Events always reflect a discrepancy between an expected state or behavior and the actual state or behavior in the application. The test defines what is expected, and these expectations are compared to the actual implementation in the application. An event can therefore be the result of one of three causes (see figure 1):

1. A mistake in the test specification (what is expected), for example a logical mis-take in the test or a misunderstanding between the test and development teams.
2. A problem in the test environment, including memory or performance issues, wrong paths for the environment, problems arising from other programs on the machine, database problems and runtime environment issues.
3. An actual error in the software; these are the errors that tests are supposed to find.

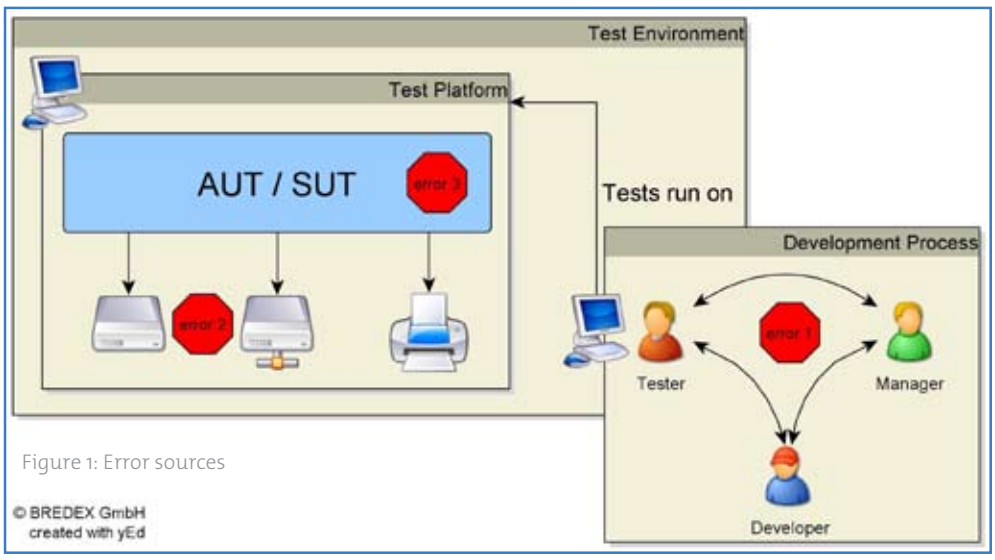


Figure 1: Error sources

Minimizing errors that occur because of these first two factors is possible by adapting the test process and by setting up a stable environment (more on this later). Neverthe-less, regardless of the cause of the event, it is important to be able to react accordingly, depending on the type, location and severity of the event.

Events can manifest themselves in various ways in an automated acceptance test:

1. The expected state does not match the actual state:
    a. The data used in the test are not valid in the context of the application (for example, a menu path that the test tries to select doesn't exist).
    b. An assertion made by the test fails during execution (for example, a result calculated by the application).
2. The expected behavior and the actual behavior are different:
    a. The workflows defined in the test cannot be executed on the application (e.g. the test assumes that the workflow takes place in one dialog, but the workflow is divided over multiple dialogs in the application).
    b. The component in the software which should be tested cannot be found at runtime (this can either be a result of a different expected workflow or an unexpected event in the software).

Knowing what types of event to expect during test automation can help to plan and adapt the event handling system, so that maximal test coverage is attained despite any events which occur. This being said, preparing for unexpected events can be a challenge.

Planning for unexpected events

In any test, there are theoretically a number of things that could go wrong. Buttons that need to be clicked could be disabled; whole screens could be blocked by an error message and so on. Depending on the use case being tested, there could be some time distance (usually 1-3 actions) between an event occurring and this event adversely affecting the test. This makes searching for the cause of the event (and handling it, if possible) more difficult. Ideally, the test should be structured so that events are discovered as quickly as possible. In the example of the button, an enablement check (expected value: true) would suffice to prevent the event before a click has even been attempted. In other words, checks can be used to identify an upcoming event (a falsely disabled component) instead of producing an event later on in the test (the click was sent, but the dialog didn't close and the next action fails due to the dialog still being visible).

In the strictest case, this would mean performing checks before every single action. This does, however, require careful planning in terms of test design to maximize reusability and minimize effort. The advantage of this approach is that the events produced are more likely to be exceptions, which can be dynamically handled in the test (the test can react to a failed check and create the state or condition necessary to continue, much as a manual tester would do). Alongside this aspect of test design, a good structure of the test as a whole is critical to be able to cope with events.

Test design to support event handling: independent use cases

Designing tests so that individual use cases are independent of each other is a critical prerequisite for effective event handling. A test can only meaningfully continue after an event in one use case if the following use cases do not assume that this first use case was successful.

This means dividing tests into separately executable use cases, which have their own structure:

- SetUp: This ensures that the application is in the expected state for the use case to begin. The starting point for any use case is often a freshly started ap-plication. After making sure that the application is ready to be tested, the SetUp can then continue to create the conditions required by the application to run the use case.
- Workflow: The whole workflow for the use case to be tested should follow the SetUp. The use case should have no prerequisites from any other use cases and should create no dependencies for further use cases.
- TearDown: Once the use case is completed, the TearDown brings the applica-tion back to the state expected as a starting point for the SetUp. This can in-volve removing any data created during the use case and recreating the default interface for the application.

Within the whole test, this structure results in a "handshake" between each use case. If one use case was successful, the application is already in the correct to state to begin with the second use case.

One question arising from this structure is how to manage data or preconditions required to execute a use case. In a use case to delete a project, the SetUp should ideally prepare the project to be deleted – although preferably not via the test automa-tion itself (i.e. do not write tests to create necessary data for other tests). Specifying tests which set up data is time-consuming (both in terms of test creation and test execution) and creates dependencies in the use case (if there is an error creating the project in the application, the delete action cannot be tested). In such cases, a concept similar to mocks and stubs is necessary. For applications which use a database, for example, a set of default data could be present in the test database. This data can then be recreated in the database using a simple script when necessary.

This being said, the paradigm create – edit – delete is common in tests and, depending on the project, it may not be desirable to separate all three use cases. In such situa-tions, it is important to be aware of the dependencies in a use case and to minimize their effects where possible.

Test design: handling events at the right level

Having designed automated tests as independently executable use cases, event handling can be added. There are two levels where it is reasonable to deal with events – locally and globally (see figure 2):

Local event handling can be used for exceptions which are expected (known prob-lems or inconsistencies) and which can be resolved or ignored for the use case. The aim is to document the error, to dynamically "fix" it for the test and then to continue. For example: a check-box should be activated for the test to succeed. If the check-box is not activated, the error handling can activate it so that the test can continue.

Global event handling is used for the "unexpected" errors which are neither antic-ipated nor "fixable" in the test. Such errors usually mean that the application is in an inconsistent or unknown state and therefore that the rest of the use case could be affected by inherited errors. The aim of global error handling is to document the error (screenshot, description of the error type and the use case it occurred in, log entry) and then to prepare the application for the next use case. By using the same TearDown function as at the end of a successful use case, the same handshake between use cases is assured. The success of such global handling rests on the independence of the use cases – if the following use case assumes the successful completion of this failed use case, then more events will follow.

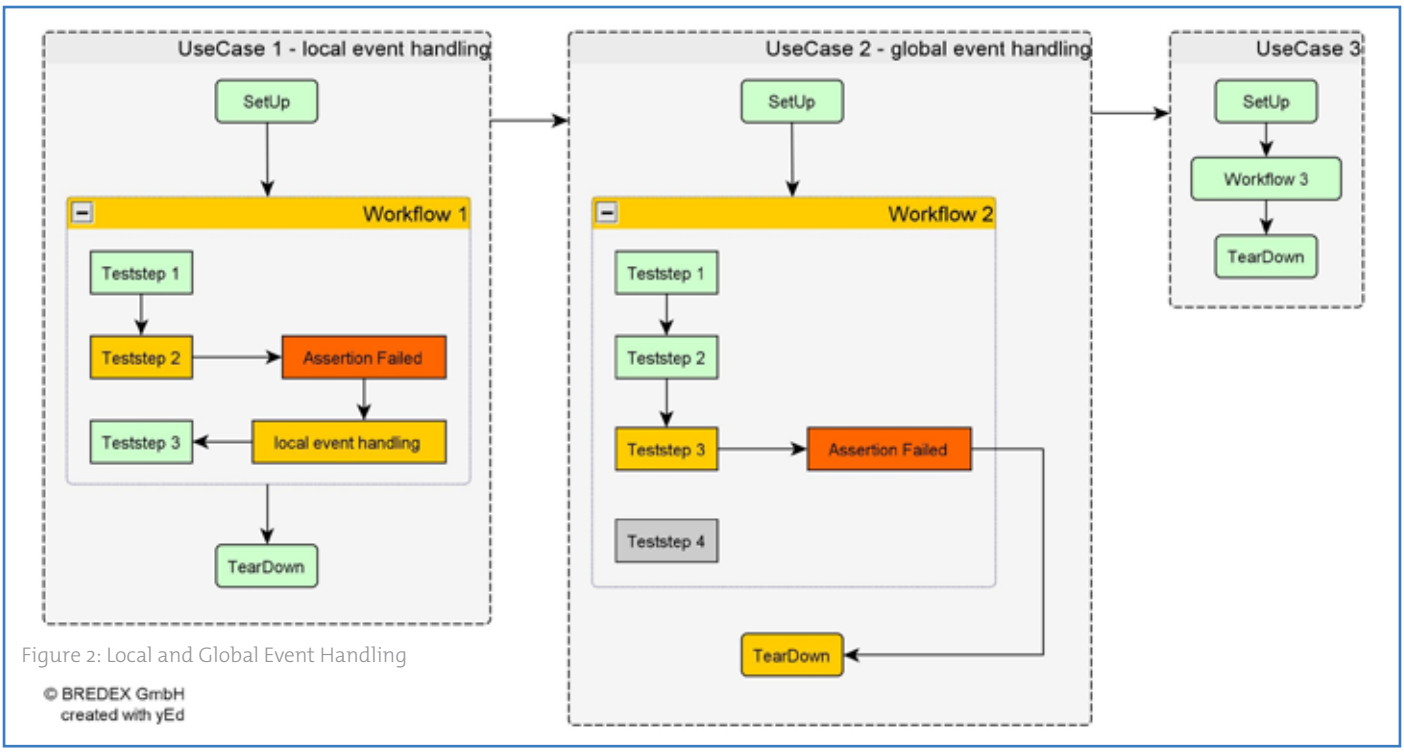


Figure 2: Local and Global Event Handling

It can be tempting to try to react to every potential event locally; however, a good rule of thumb is to start with global handling (which ensures that all events are caught and that the test can continue) and to add local handling for manageable exceptions as it becomes necessary.

Can events be avoided altogether?

As mentioned above, events can come from various factors. Events resulting from actual errors in the software are unavoidable (if they were avoidable, there would be no need to test!), but events that come from the other two areas (from the tester and from the environment) can often be reduced. Events resulting from mistakes in the test specification can often be avoided by improving the communication in the test and development process. By inviting the test team to development meetings and by formulating requirements directly in terms of testable criteria, many misunderstandings can be discussed and the negative effect on the tests avoided. Encouraging the test and development teams to communicate changes and questions outside of formal meetings can also lead to fewer events from this area.

Events that occur because of the environment can be minimized by having a robust strategy for preparing necessary data and conditions for the test without using the test itself to set them up. A robust synchronization within the tests (flexibility if certain actions or processes take longer than usual) can also reduce the amount of unnecessary disruptions to the test. As well as synchronization, the tests also need to include as much "intelligence" as a manual tester would use to react to small changes or inconsistencies, plus an explicit understanding of how the application's interface and logic works (how focus works in the application, how the completion of processes is displayed in the interface etc.). In this way, tests do not rest on assumptions that may change or that turn out to be false, but are based on well-defined workflows and actions.

Dealing with errors in a project

Handling events in the tests is an important step, but once an error in the software has been discovered, there must be processes in place to handle these errors within the team. There are two main options for dealing with errors found by the automated tests:

1. Fix the error in the software instantly. This has the advantage that the fix is close to the development, which reduces the cost to resolve the error and also im-proves the chance that the developer who introduced the error is still working on the code. It also means that the affected use case is up and running in the shortest time possible so that it can continue to test the software as intended. Because the time between finding and fixing the error is so short, there is no need to consider removing the test from the set of productive tests. The disadvantage of this approach is that the team can end up doing "error driven development", whereby normal tasks are left to one side to ensure that the automated tests are kept running. This approach is also unsuitable for errors that are not quick and easy to fix, but which are just the "tip of the iceberg".
2. Write a ticket for the error. This has the advantage that the resolution for the error can be planned better (which is useful if the fix has wider-reaching conse-quences or risks in the software). If a ticket has been written, it may be worth considering removing the affected use case from the productive tests and instead running it with "known error" tests (which is easy to do if the use cases are independent of each other). The disadvantage of this approach is an accumulation of tickets over time, while parts of the affected use cases cannot run until the error is resolved.

Conclusion

Event handling isn't necessarily the first thought when it comes to writing automated tests. However, as the amount of tests grows, and the reliance of the team on the quality monitoring increases, a well-thought out event handling mechanism can make the difference between comprehensive test results despite errors and a complete test fall-out after every single misunderstanding or event.

The implication of this is that the test tool used to automate the acceptance tests must support the mechanisms required to implement successful event handling. This includes being able to react dynamically to events in a variety of ways depending on the event type, level and the situation where the event occurs. It also means that tests should be under continuous scrutiny so that improvements to the test structure and additional local event handling can be added as necessary.

Biography

Alexandra earned a degree and an MA in linguistics from York University before starting work at BREDEX GmbH. She is actively involved in various roles at the company – alongside dealing with customer communication, support, training and demonstrations, she also enjoys representing the customer perspective in the software development process. She has experience of working and testing in both traditional and agile projects.

Markus is a software developer and tester at BREDEX GmbH. His main areas of expertise are the design and maintenance of automated tests and Eclipse RCP development. He is one of the core team developers for the automated test tool GUIdancer and holds a Diplom (German degree) in Computer Science from the University of Applied Sciences in Braunschweig.

Test automation as technique to leverage added value of portability testing

by Wim Demey

Of all test types, portability testing is not the most frequent one being performed. However, if you have to support multiple configurations, you cannot bypass portability testing.

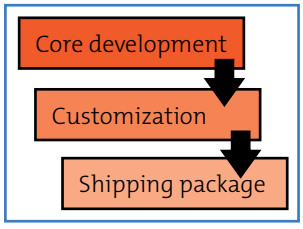
In most cases portability testing is just a subset of the functional tests that are redone on the supported configurations. But what about the installation of packages? Have you thought about the impact of installation errors may have on the perception and trust of end-users of your software? This is definitely a hidden area, where portability testing could bring some added value. Furthermore, using test automation tackles the problem of having all hardware configurations and resources available. For this reason, I have worked out a promising solution to leverage the value of portability testing.

1. Context

For two years the idea has developed to explore the possibilities of test automation for portability testing. At the time when it started, I was managing a team which was only responsible for doing manual portability testing of software packages (.msi file). This was the last quality gate before the package was shipped towards the customers. Due to complaints about failing installations, this last test had been added to the quality process.

Based on a checklist, the team went through the installation wizard step-by-step and focused especially on syntactic items (e.g. display of correct version numbers, spelling mistakes). This consumed a lot of time, and the tests were very repetitive.

The fictive case below illustrates very well what we were doing.



The company "Soccer is my passion" develops web-based software to plan the calendar and manage standings of national football leagues all over the world.

Some core modules (e.g. team management, calendar planning) are developed centrally by the company.

Per region (Asia, Africa, EMEA, ...), another team is responsible to adapt the core modules to the needs of each interested association (e.g. number of teams, play-offs/relegation, the way of calculating points in case of won, lost, draw).

Finally, they build an installation package (.msi or .exe file) and ship it to the interested association.

Moreover, the software must support a variety of operating systems (e.g. Linux, Windows XP, Windows Vista and Windows 7), browser versions (Internet Explorer 7 & 8, Firefox) and database versions (Oracle 9, 10 and SQL2005, SQL2008).

2. Portability testing - love or hate it

In our example, we have different configuration parameters to take into account:

- Operating systems: 4 types
- Browser versions: 3 types
- Database versions: 4 types

Testing all possible combinations by applying the multiple condition coverage technique leads to a huge number of test cases: 4 x 3 x 4 = 48 cases.

Even if you can still reduce these numbers by applying techniques like orthogonal arrays or pairwise testing, it remains a lot of testing. In our team we listed all combinations and went through them with the requesting stakeholder, who indicated the most critical/important ones to be tested.

Thereafter, the team started the testing, which sometimes lasted many days with several resources. It is understandable that the huge effort, regardless of the question of having all required test configurations, is a reason for skipping or lowering the priority of portability testing.

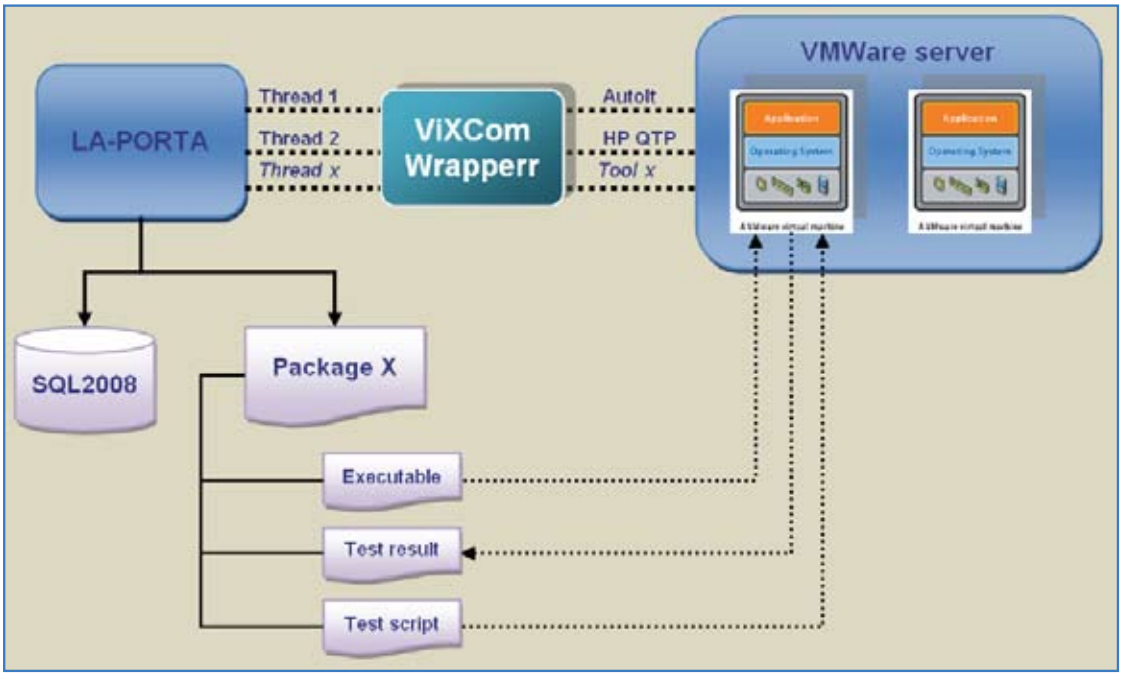
Besides this reason, there are some other facts which explain why portability testing is unpopular:

- Portability testing is considered as a duplication of regression testing and hence repetitive and not challenging (enough) for testers. Testers risk losing their focus when they run portability tests time after time.
- Portability testing is positioned late in the V-model and is consequently the first candidate to be dropped when a project and especially testing are under time pressure. Therefore it is recommended to think about it in earlier phases.
- A bottleneck is often caused by the availability of all hardware configurations. Besides the infrastructure costs, this means a lot of workload to prepare and support all required test environments. Needless to say that virtualization tackles this issue. With virtualization hardware savings can be made and operational efficiency increased.

3. Combination of both worlds as a solution

With this context in mind, I looked for a solution which combines test automation and virtualization, in order to extend the added value of portability testing. Personally, I believe that testers could hold out their hands towards infrastructure people by testing the installation process of packages. So portability testing goes beyond its rather functional focus. Most companies have mechanisms for distributing ("pushing") software to workstations on request. Eventual problems caused by bad installation are noticed by the end-users, which can cause a lot of frustration. It would be a good idea to run a sanity check during and after the installation.

I want to emphasize that this solution is still in an experimental stage, but the first tests were promising.



The basis for my solution (which I called "LA-PORTA" (Launch Automated Portability Testing) are packages.

A package is rather generic and represents a certain version of an application that you want to test on one or more configurations (= test run). It does not matter if the package is a custom- development or third-party software.

The test run contains all information about the virtual configuration (OS, database, ...). If necessary, you can even add the installation mode (install, remove, repair) as a parameter.

Once a test run has been defined, it is added to the test set, where you can select which test run(s) will eventually be executed.

During the implementation of this concept, I was also wondering if there were free tools or solutions available, which could be integrated in my solution "LA-PORTA" (Launch Automated Portability Testing).

From an architectural point of view, LA-PORTA exists of 5 components, which I will later explain in detail:

- Custom application (= LA-PORTA)
- Virtualization product
- API, which enables automated interaction between custom application and virtualization product
- File structure
- Test automation tool

3.1. LA-PORTA

This is a lean application developed in .NET. and has three main functionalities:

- ***Configuration*** allows you to manage parameters like available databases, operating systems and other values used to fill drop-down lists.

  Besides the parameters, you can manage the virtual images representing the configurations on which you test and the packages (e.g. Windows XP with SQL2005, Windows XP with Oracle9)

- Through a ***test run***, you link the package to a certain virtual image. In addition, you can also define the pre and post conditions. These are especially important if you want to re-use the virtual image and keep it as clean as possible (e.g. by removing all copied folders and the installed package).

- Finally, in ***test set (=sub-menu of test run)*** you select the test run(s) you want to execute. During test execution, a log is updated in real-time, which can be saved.

3.2. Virtualization product

In my case, I have experimented with VMWare Server, which is completely free. The disadvantage of this solution is the bigger footprint, because it installs on top of the underlying operating system. The VM family has other products like ESX/ESXi, which are hypervisors and rely on a built-in operating system to perform core virtualization tasks.

On the VMWare server, several VMware images are hosted, each representing a certain configuration.

3.3. VIX COM

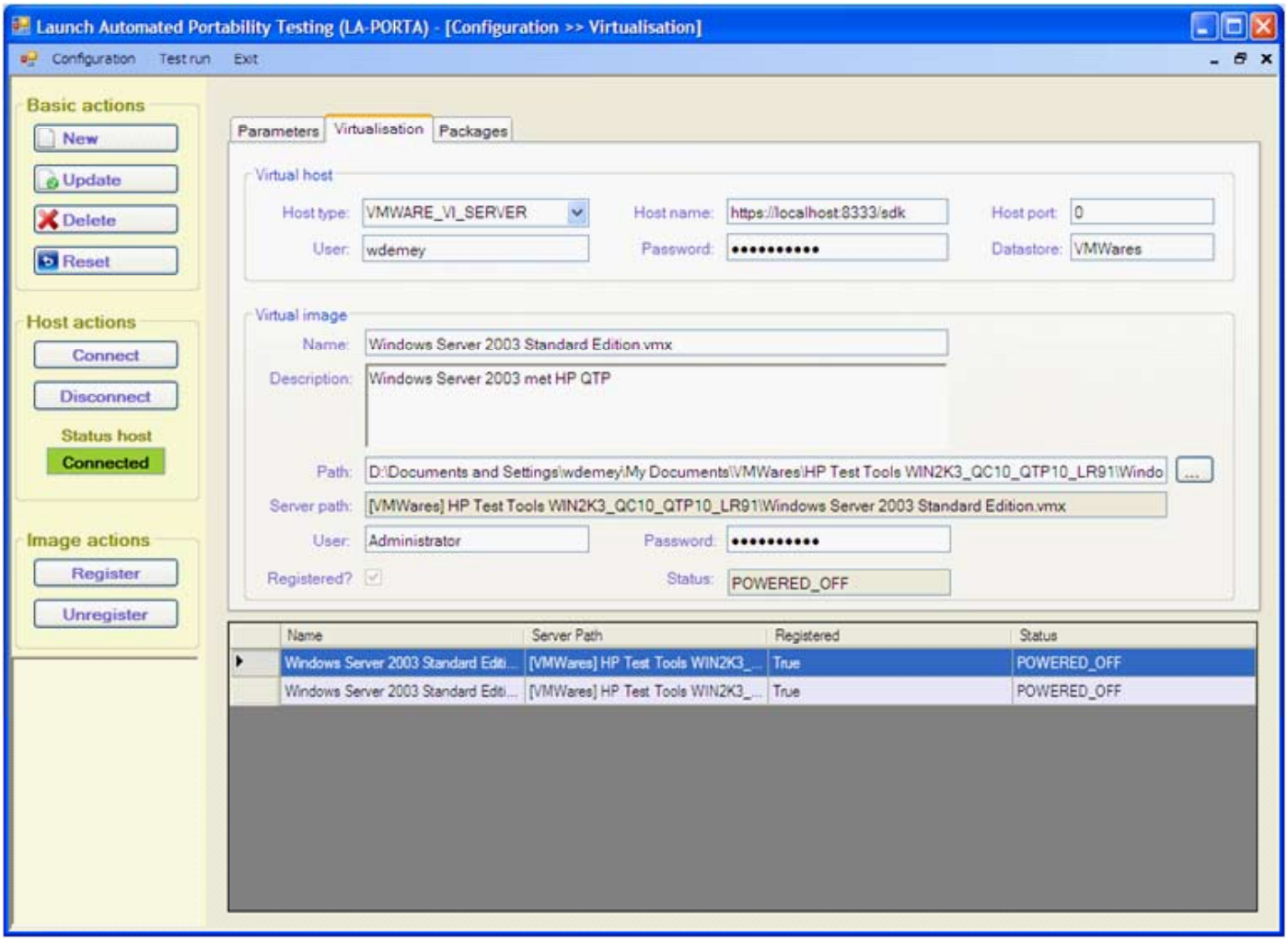
The key to success of automation is the possibility to automate all virtualization-related tasks (e.g. start/stop server, create snapshots, log in/log off, suspend/resume image).

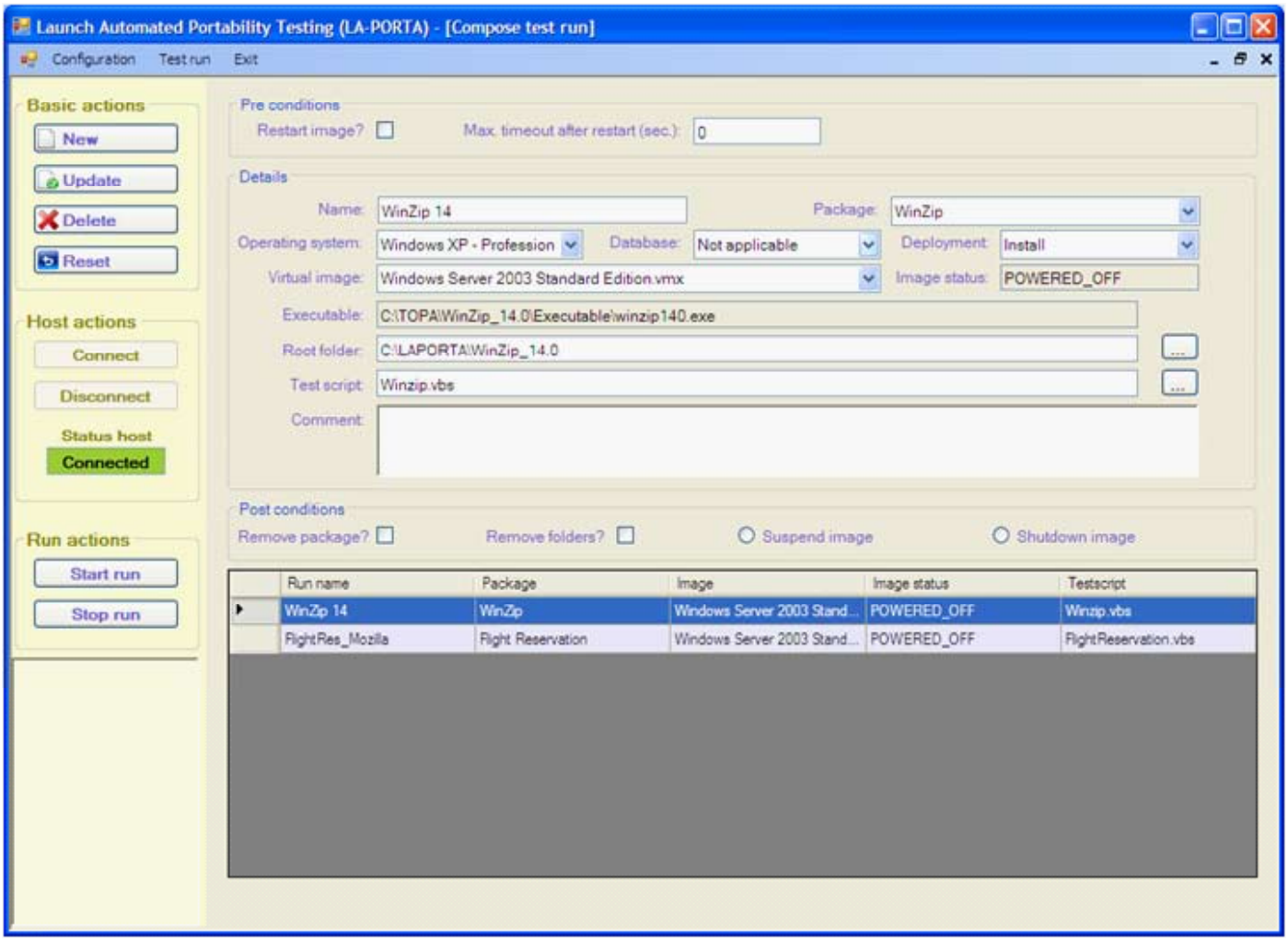
The great advantage of using VMWare is the availability of a well-documented VIX API (written in C#), ease of use, and practical support for both script writers and application programmers.

To make it still easier, I have chosen the Vix COM wrapper (www.codeplex.com), which wraps the original API into a library and hence could be used in .NET.

The main benefit of this wrapper is the synchronous implementation of all functions, which hides the user from the need to use call-back functions and/or polling/blocking mechanisms.

The big challenge I had to face was getting my test runs executed in parallel. I managed to solve this by putting each run into a separate thread.

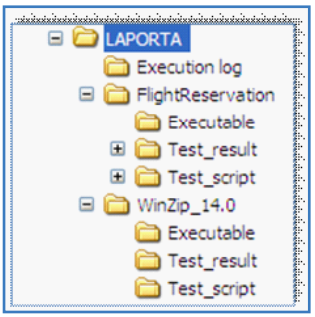

Launch Automated Portability Testing (LA-PORTA) - [Configuration >> Virtualisation]
Configuration
Test run
Exit
Basic actions
New
Update
Delete
Reset
Host actions
Connect
Disconnect
Status host
Connected
Image actions
Register
Unregister
Parameters
Virtualisation
Packages
Virtual host
Host type: VMWARE_VI_SERVER
Host name: https://localhost:8333/sdk
Host port: 0
User: wdemey
Password: ••••••••••
Datastore: VMWares
Virtual image
Name: Windows Server 2003 Standard Edition.vmx
Description: Windows Server 2003 met HP QTP
Path: D:\Documents and Settings\wdemey\My Documents\VMWares\HP Test Tools WIN2K3_QC10_QTP10_LR91\Windo
Server path: [VMWares] HP Test Tools WIN2K3_QC10_QTP10_LR91\Windows Server 2003 Standard Edition.vmx
User: Administrator
Password: ••••••••••
Registered?
Status: POWERED_OFF
Name
Server Path
Registered
Status
Windows Server 2003 Standard Editi...
[VMWares] HP Test Tools WIN2K3_...
True
POWERED_OFF
Windows Server 2003 Standard Editi...
[VMWares] HP Test Tools WIN2K3_...
True
POWERED_OFF


Launch Automated Portability Testing (LA-PORTA) - [Compose test run]
Configuration
Test run
Exit
Basic actions
New
Update
Delete
Reset
Host actions
Connect
Disconnect
Status host
Connected
Run actions
Start run
Stop run
Pre conditions
Restart image?
Max. timeout after restart (sec.): 0
Details
Name: WinZip 14
Package: WinZip
Operating system: Windows XP - Profession
Database: Not applicable
Deployment: Install
Virtual image: Windows Server 2003 Standard Edition.vmx
Image status: POWERED_OFF
Executable: C:\TOPA\WinZip_14.0\Executable\winzip140.exe
Root folder: C:\LAPORTA\WinZip_14.0
Test script: Winzip.vbs
Comment:
Post conditions
Remove package?
Remove folders?
Suspend image
Shutdown image
Run name
Package
Image
Image status
Testscript
WinZip 14
WinZip
Windows Server 2003 Stand...
POWERED_OFF
Winzip.vbs
FlightRes_Mozilla
Flight Reservation
Windows Server 2003 Stand...
POWERED_OFF
FlightReservation.vbs

3.4. File structure

A template folder structure increases the reusability of the solution and makes it independent of the test tool you use for the automated portability test.

Each package has a separate folder in which you find the same recurrent folders:



- Executable

  = contains the package/application which is subject for portability testing.

  Mostly this is a file with an extension .msi or .exe

  Optionally, the executable can be an input for the test script.

- Test_result

  = at the end of a test run, the result files (no matter whether just a text file or test tool result files) are copied back to the host.

  This folder does not only contain result files, but also an execution log.

- Test_script

  The driver script is just a simple vbs script which calls a specific test tool script. This mechanism guarantees that it doesn't matter which test tool you are using, as long as you can launch the script through a vbs file.

  Below you can see an example using AutoIT and HP QuickTestProfessional.

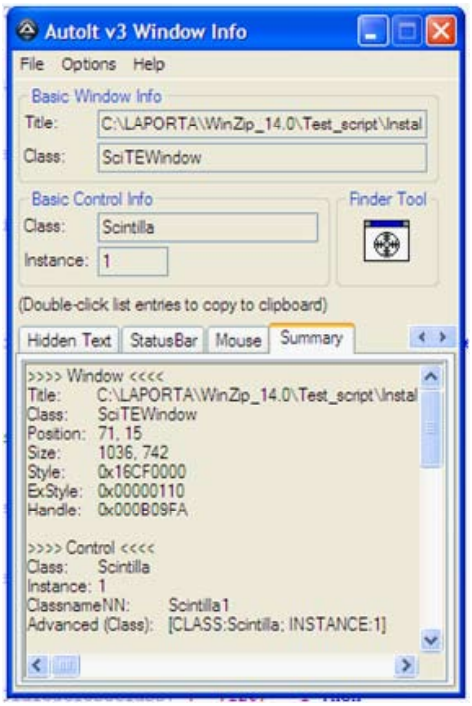
```
Option Explicit
Dim wshShell
Set wshShell = WScript.CreateObject("WScript.Shell")
wshShell.Run "C:\LAPORTA\WinZip_14.0\Test_script\Install_WinZip14.exe"
```

```
Set qtApp = CreateObject("QuickTest.Application") ' Create the Application object
qtApp.Launch ' Start QuickTest
qtApp.Visible = False' Make the QuickTest application visible

' Set QuickTest run options
qtApp.Options.Run.RunMode = "Fast"
qtApp.Options.Run.ViewResults = True

qtApp.Open "C:\LAPORTA\FlightReservation\Test_script\Web_FlightReservation_01", True ' Open the test in read-only mode

' set run settings for the test
Set qtTest = qtApp.Test
qtTest.Settings.Run.OnError = "NextStep" ' Instruct QuickTest to perform next step when error occurs

Set qtResultsOpt = CreateObject("QuickTest.RunResultsOptions") ' Create the Run Results Options object
qtResultsOpt.ResultsLocation = "C:\LAPORTA\FlightReservation\Test_result\QTPResult" ' Set the results location

qtTest.Run qtResultsOpt ' Run the test
```

3.5. Test automation tool

During my investigations, I was wondering which tools could be used to automate a portability test in a virtualized test environment. Apart from the commercial ones (of which I have used HP QuickTestProfessional), I did find an interesting open-source tool; AutoIt (http://www.autoitscript.com/autoit3/).

This freeware BASIC-like tool is designed for automating Windows GUI and general scripting. It is possible to automate keystrokes and mouse movements. Initially, the goal was to automate the PC roll-out of packages.



Besides the fact that it is very simple and easy to learn, it has some important benefits:

- The tool contains a freeware script editor with built-in function libraries (e.g. open and write to a text file).
- The generated script can be compiled into stand-alone executables (no need for agents to launch AutoIT scripts).
- Just like the commercial tools, it has a spy function, which captures the necessary information about object properties.

Using this tool, I was able to use automation to unpack the msi installation wizard, step by step, while I did some checks on titles in Windows etc.

To be honest, I experienced that the commercial tool was better to handle complex installation wizards or applications.

4. Conclusions

Although portability testing is not performed often, it is definitely an interesting test type, not only to check whether your package works properly on different configurations, but also to check the installation process itself. This could prevent a lot of frustration for end-users.

So, portability testing can be used in a broader context than just functional testing..

Using virtualization solutions solves the problem of not having all platforms available. Moreover, the well-documented API allows you to integrate it into your own custom application. In this way, fully automated portability testing becomes a reality!

Biography

Wim is an ISEB & TMap certified test consultant working for CTG Belgium with over 13 years of test experience in several areas: telecom, social security, banking, utilities, validation.
In his assignments he has held different roles (e.g. test engineer, test lead, test manager, training/coaching, pre-sales).
Wim has a special interest in test tools and has performed several assignments in this area.
In recent projects, he also focused on pure infrastructure-related projects (workstation upgrades, split of back-end infrastructure, ...).
Finally, Wim is a regular speaker at national/international test conferences/seminars (KVIV, TestNet, Eurostar, Dutch Testing Conference, SEETEST, Conquest, DIA, SSQS).

iqnite 2010 Germany

27 – 30 April 2010 | Dusseldorf

Meet experienced industry insiders and eminent experts at the **iqnite** conference at Dusseldorf.

Register now and discuss current standards and trends as part of the software quality community.

iqnite-conferences.com/de

On the following pages, please find a selection of our renowned **iqnite** exhibitors.

C1 SetCon

Solange Sie sich in der Testphase befinden, können Überraschungen noch amüsant sein

Professionelles Testen gibt Ihnen die Sicherheit, Fehler und Performance-Probleme rechtzeitig zu entdecken

Wir professionalisieren Ihren Softwaretest und optimieren die Performance Ihrer Applikationen.

Unsere Leistungen:

- Testreifegradermittlung
- Testprozess- und Teststrategieberatung
- Testmethodikberatung
- Testmanagement
- Testkonzeption und -durchführung

Wir vereinfachen Ihren Last- und Performancetest mit TAPE:
TAPE ist das einzigartige Tool von C1 SetCon zum Management Ihrer Last- und Performancetest-Prozesse.

www.tape-loadtest.com

Kontakt Zentrale München: Kapuzinerstraße 7a · 80337 München · Tel: +49(0)89 589399-0 · info@C1-SetCon.de · www.C1-SetCon.de

Assess your Global Sourcing capability with Testing

Outsourcing as a concept has been in existence for quite long; the idea being organisations tend to become more effective and grow fiercely when they focus on their core competences and leave the rest for experts to handle. When organisations grow, their IT organisations tend to transform into cost centers. Overheads get built into the system and, if unaddressed, tend to wipe off margins. Outsourcing of non-core activities to experts will help attainment of quality solutions at optimised cost. Outsourcing is a strategic initiative, and is about partnering. It involves identification of tasks that can be outsourced, evaluation and selection of vendors, identification of risks involved, mitigation plans, contracts planning and sourcing, continuous evaluation and handling the right portfolio mix. The choice of vendors is the most critical activity as the entire ecosystem comprising of stakeholders will be impacted by the move and performance. Critical parameters such as vendor capability to deliver solutions on time, credibility, reliability, mode of engagement etc. are to be evaluated before the decision to engage is opted. In an ideal scenario, a vendor can transition from a mere service provider to a strategic partner. These symbiotic associations have always enabled faster growth of the partnering organisations. However, there are factors that can dent relationships which may be internal or external to the ecosystem. Overlooking these factors can adversely impact the entire move to outsource, reaping little or no return on investment.

Why Outsource Testing

Software testing involves validation of the application to ensure that the user experience is in keeping with the specifications. Today, companies like Cognizant offer testing services on a standalone basis, focusing on supporting enterprises' and product companies' testing needs through Independent Verification and Validation (IVV) of their software. Customers tend to outsource testing because of the following reasons:

- This helps manage resource requirements based on need, especially while catering for the sudden rise and fall in demand.
- Industry best practices and defined methodologies help in achieving higher quality.
- The availability of testing tools expertise and infrastructure ensures reduced cost, improved quality and increased speed.
- Continuous process improvements result in higher productivity and increased cost savings.
- Independent validation helps boost confidence levels during application go-live decision-making.

Outsourced testing models provide the benefits expected in any outsourced IT service model, such as access to a large pool of dedicated professionals with a wide range of skills, and increased efficiencies due to the value creators and solution accelerators developed based on the teams' deep expertise and experience.

Distinct advantages of outsourcing software testing

By establishing the right enterprise IT strategy and blending in QA smoothly many of the prevalent gaps can be filled. The decision makers have to evaluate ways to build efficiencies into their delivery processes leading to cost optimisation and delivery predictability. Organisations taking this transformation trajectory need to continuously focus on identifying gaps in the current state and bridge them through innovative means. Hence the first step to this exercise is to assess the current operating state of your QA organisation and identify zones where value can be unbundled.

Testing application functionality requires good understanding of the business domain and hence organisations used business analysts or subject matter experts to validate the application's suitability to business. The emergence of software testing service providers helped business reap the following benefits:

- Cost optimisation through efficient resourcing and utilisation.
- Capacity to address sudden spike and drop in demand, thereby eliminate overheads.
- Manage resource requirements based on need; especially while catering for the sudden demand rise & fall.
- Continuous process improvements result in higher productivity and in increased cost savings.

Testing service providers invest significantly in various avenues of testing such as infrastructure, processes, people, tools, research, training and development etc. This helps their clients obtain higher value for the cost incurred, thereby maximising the return on outsourcing.

Organisations most likely to outsource testing

- Organisations experiencing increased quality issues owing to insufficient testing or ineffective processes and techniques
- Organisations witnessing rising costs related to quality. This may be due to too much effort spent on testing or may be due to business analysts and SMEs performing testing. Also, organisations that have built a huge mass of contractors over a period of time for testing may prefer to opt for outsourced testing as a measure to reduce cost.
- In the case of large applications, typically enterprise implementations, the support from vendors would cease after implementation and stabilisation. In such scenarios, the client tends to look at testing service providers instead of building an internal QA team.
- Organisations looking to optimise cost and improve coverage through automation. For them, building a test automation team internally may be a costlier option.

Sumithra Gomatam,
Head Testing Services, Cognizant Technology Solutions

About the Author

Sumithra is the Business Leader of Cognizant's Testing Services practice and the Indian Business Unit of Cognizant. She has over 22 years of experience in the IT industry and joint Cognizant in 1995. Today, with over 10600+ career testers, Cognizant's testing practice is well acclaimed in the industry and has established itself as one of the fastest growing practices in Cognizant.

About Cognizant

Cognizant is a leading provider of information technology, consulting, and business process outsourcing services. With over 50 global delivery centers and more than 68,000 employees as of September 30, 2009, we combine a unique onsite/offshore delivery model infused by a distinct culture of customer satisfaction.

Cognizant™
Passion for building stronger businesses

Cognizant Germany, Email: sales.de@cognizant.com
Torhaus Westhafen, Speicherstrasse 57-59, 60327 Frankfurt am Main
Phone +49 69 2722 695-0, Fax +49 69 2722 695-11
Visit us online at www.cognizant.com.

Best practices for implementing automated functional testing solutions

There is no question that rigorous functional testing is critical to successful application development. By automating key elements of functional testing, companies can meet aggressive release schedules, test more thoroughly and reliably. But automating functional testing raises some questions:

- What are the costs of automating the testing processes and what is the ROI?
- Which applications/processes are candidates for automated testing?
- Will new training be required and will that impact current project schedules?
- What is the proper methodology for automating a testing effort?

How to identify candidates for automated testing

In general, it makes sense to focus automation efforts on critical business processes and complex applications. But if multiple software testers spend many hours per day on software quality, companies definitely benefit from moving to automated testing. It will generate a positive return if the application requires regression testing, requires multiple builds/patches/fixes, needs to be tested on numerous hardware or software configurations, and supports many concurrent users. In addition, if repetitive tasks such as data loading and system configuration are involved, or if the application needs to meet a specific service-level agreement (SLA), automation will certainly make economic sense. Automation does not make sense when testing the usability of a user interface (UI) or for exploratory testing or applications that are not yet mature. When it comes to automating functional testing, the best approach is to determine with as much precision as possible what the hard-dollar costs are and then compare them to the hard- and soft-dollar benefits of automating the test efforts. Hard-dollar costs to consider in an ROI calculation include acquisition costs, hardware costs, labor costs and training costs.

Evaluating automated testing software: What to look for

Scriptless representation of automated test

The testing product should offer a point-and-click interface for interacting with the application components under test. Testers should be able to visualize each step of the business process and view and edit test cases intuitively

Integrated data tables

One of the key benefits of automating functional testing is the ability to pump large volumes of data through the system quickly. But it is also important to be able to manipulate the data sets, perform calculations, and quickly create hundreds of test iterations and permutations The products should offer integrated spreadsheets with powerful calculation capabilities

Clear, concise reporting

Products should automatically generate reports that display all aspects of the test run and the results. The reports should provide specifics about application failures occurred and what test data was used; present application screen shots for every step; and provide detailed explanations of each checkpoint pass and failure.

THE COMPLETE SOLUTION FOR SOFTWARE TESTING

Does your testing occur too late in the development lifecycle?

Does your application quality suffer from incomplete requirements?

Is manual testing delaying product releases?

Are you struggling to manage testing as you move to Agile development?

Micro Focus Testing solutions help customers incorporate quality into software delivery from the beginning of the development lifecycle. Whether you are operating in a traditional environment, looking to transition to Agile or working across a mix of methodologies, Silk Testing solutions help align your business requirements and quality expectations. Covering regression, functional, performance and load testing processes, Silk empowers you to reduce business risk, ensure the deployment of high quality IT projects and maximize your ROI.

Visit our booth to learn how you can make quality a core value of your software application delivery. Alternatively visit our website www.microfocus.com.

Borland®
(A MICRO FOCUS COMPANY)

MICRO FOCUS®
Leading the Evolution™

Load Testing for all your web applications

NeoLoad

Test your web application's performance easily and deploy with confidence!

NeoLoad, load testing solution, ensures better efficiency to perform your tests faster, while providing pertinent analyses and full support for all new technologies.

Test your Web application's performance easily and ensure trouble-free deployment thanks to NeoLoad!

- HTTP
- AJAX & RIA
- Adobe Flex
- Silverlight
- Oracle Forms
- SOAP
- GWT

NEOTYS
Make sure it works!

More details and free trial on
www.neotys.com

Visit us at iqnite GERMANY2010

Basel | Genève | Freiburg

SYNSPACE
Die Prozessmanufaktur

Become an ISTQB® Certified Tester.
With SynSpace!

Being an international training company, SynSpace offers training courses for ISTQB® Certified Testers, both at foundation and advanced levels. Accredited SynSpace trainers with several years of experience in the IT sector will tutor you for the final exam and teach you how to practically apply your knowledge in the professional world.

Basic and further training are structured into two consecutive steps:

- **Foundation Level:** conveys a broad overview of tasks, methods and techniques involved in software testing.
- **Advanced Level:** immerses specific topics and offers highly practical courses that will teach you to professionally handle demands and challenges in practice.

Within the scope of the ISTQB® Certified Tester Standards (Advanced Level), you may choose the type of training plan that best suits your individual task-related requirements and will lead you to your goal, in accordance with the latest curriculum:

- **Test Manager**
- **Test Analyst**
- **Technical Test Analyst**

For those in a hurry

e.g. Test Manger training
1 x 5 days compact training

For the flexible ones

Different modules can be combined, e.g.
1 day basics training
2 days further training
2 days final training

iqnite GERMANY2010

Visit us on the internet.
Or at the iqnite.
We are looking forward to seeing you at the SynSpace booth at the iqnite conference from 28. to 30. April 2010 in Düsseldorf. We will gladly explain to you how SynSpace can help to effectively increase your company's testing know-how.

SynSpace AG | SynSpace SA | SynSpace GmbH

Michael Landwehr | Director Sales & Marketing
phone +49 761 476 45 65 | fax +49 761 476 45 68
info@synspace.com | www.synspace.com

Zielsicher zum Erfolg

Trainieren Sie Qualität

- Praxistage zu jedem Seminar aus der ISTQB® Certified Tester Reihe NEU
- Testen in Migrationsprojekten NEU
- SOA – Qualitätssicherung von Service-orientierten Architekturen
- Agile Software-Entwicklungsmethoden und -Tests
- und viele weitere

Alle SQS Seminare unter **www.sqs.de/training**

Anmeldung über **seminare2010@sqs.de**

SQS Software Quality Systems AG
Stollwerckstraße 11 | 51149 Köln | Telefon 02203 9154-17 | Telefax 02203 9154-15

SQS
SOFTWARE QUALITY SYSTEMS

ISTQB

German Testing Board

www.german-testing-board.info

ISTQB® Certified Tester

Über 125.000 Certified Tester weltweit. Wann gehören Sie dazu?

Manche Ausbildungen öffnen Wege, andere eine ganze Welt.

Die Schulung zum ISTQB® Certified Tester führt zu einem weltweit anerkannten und etablierten Zertifikat. Rund 14.000 Fachkräfte alleine in Deutschland haben es schon. Und Zehntausende in weiteren 47 Ländern - von A wie Amerika bis V wie Vietnam.

- Der ISTQB® Certified Tester ermöglicht Karrieren - mit ihm liegt erstmals ein internationales und branchenübergreifendes Berufsprofil für Software-Tester vor.
- Der ISTQB® Certified Tester macht die Arbeit leichter - Tester sprechen nun die gleiche Fachsprache, benutzen die gleichen Begriffe.
- Der ISTQB® Certified Tester hilft Kosten zu senken - Durch geschulte Tester werden Fehler bereits in frühen Phasen der SW-Entwicklung entdeckt.

Anmelden ist einfach.

Ein akkreditierter Trainingsanbieter ist sicher auch in Ihrer Nähe:

GTB Premium Partner

- ALTRAN GmbH & Co. KG
- EXCO GmbH
- imbus AG
- IT-Testing.de
- Knowledge Department GmbH & Co. KG
- Logica Deutschland GmbH & Co. KG
- MaibornWolff et al GmbH
- Method Park Software AG
- oose Innovative Informatik GmbH
- sepp.med gmbh
- Sogeti Deutschland GmbH
- SQS AG

autorisierte Zertifizierungsstellen

DLGI - Dienstleistungsgesellschaft für Informatik mbH

iSQI - International Software Quality Institute GmbH

Alle Trainingsprovider siehe www.german-testing-board.info

Staged Acceptance, the SOA Test process

by Chris Schotanus

Reduction of complexity, improvement of maintainability, reduction of lead time through parallel development, reuse, flexibility and so on are often mentioned as reasons for the application of SOA based systems. The ultimate business goal is to achieve high-quality systems within the shortest possible time. However, the advantages of service architecture are lost if the quality of the services is unsatisfactory or if the organization is not equipped for working in a SOA environment. Testing of services is therefore of crucial importance and will have repercussions on the organization of system development. Regarding governance, maintenance and operations this results in a different approach to acceptance of services and systems. This article gives answers to the questions of whether SOA services are best tested separately or in relation with other services, who must test in cases where one or more services are modified, and who finally accepts the systems.

The reasons for applying SOA are reduction of complexity, improved maintainability, shortening of lead time through parallel development, reuse, flexibility and so on. The advantages are lost if the quality of the services is unsatisfactory or when the organization is not equipped for working in a SOA environment. Testing of the various deliverables of a project demands special attention and has repercussions on the organization of system development. By early planning of structured test activities and by developing well documented, reusable tests, the demand for high Quality-to-Market and short Time-to-Market can be satisfied.

To the end-user of a system, the architecture is not of any importance. He sees the system from his own perspective: the outside, in many cases the user interface. The user will therefore focus on the final functionality when accepting new or changed applications.

From a *governance, maintenance and operations* point of view this is different. These disciplines demand a specific approach to testing of SOA based systems. In practice, we will see that during testing and acceptance of SOA based applications the following questions are asked:

- Do we always have to test SOA services[1] in a chain or will isolated testing of these services do?
- Who will accept the SOA based applications, and who accepts the individual services?
- Who must test if one of the services has been changed?

1 In this article a "service" or "SOA service" means a piece of software that performs a clearly defined function and that is independent of the status of other services.

fig. 1 The RRBT test management model

1. The Risk & Requirement Based Test Process

Before I can answer these questions, I need to establish how testing is done (without elaborating on this too much). During any - not only during a SOA based - system development process, testing consists of different activities that can be grouped into test project preparation and test project execution (Pinkster, 2004). The activities that belong to test project preparation are, amongst others, the product risk analysis, the creation of the test strategy and test plans, estimation and setting up the test organization. Test project execution consists of reviews and inspections of the source documents (static testing), design and execution of test cases (dynamic testing), issue management and reporting back of the results. A test manager manages all activities, and in order to bring the test process under control, it is divided into smaller phases; the test levels. This approach to testing is known as Risk & Requirement Based Testing.

2. Product Risks, Requirements and Acceptance Criteria

The requirements and product risks are the basis of the test strategy. That is why they must be defined as early as possible. And of course they must be SMART (**S**pecific, **M**easurable, **A**ttainable, **R**ealistic and **T**ime bound). This applies to both functional and non-functional requirements. Unlike frequently assumed, non-functional requirements to an application - like performance, load and stress - can be tested early in the system development project by estimating the behavior of individual services during low-level tests. However, most of the time final judgement can only be made during execution of tests in a production-like environment.

There are various possibilities to assess the non-functional requirements of a service at an early stage, like execution of an architectural review or gathering the right information (e.g. behavior regarding interoperability, load and stress) during test execution. During the review, the emphasis is on the architecture standards; the way services are programmed has influence on the behavior of the component during execution. Based on the information on beha-vior during these early tests, we can predict the expected behavior of the application in the target environment. Of course, depending on the identified product risks, it may be decided to perform a full load, stress and performance test just before going live.

3. Staged Acceptance

In a SOA environment the development process can be divided into two main processes: the application develop-ment process and the service development process. The application development process basically implies the as-sembly of existing services to an application. During the service development process, based on requirements that are defined during the application development process, services are developed that will perform certain activities more or less independently.

The service repository contains all existing SOA services and related information like the functionality, capacity, conditions for use of the services and, of course, the service owner. Since services are related to clearly identified business processes, the service owner is a representative of the business.

Especially when an organization starts to develop SOA based applications, it can occur that services are not yet available in the repository, or that existing services do not comply with the new requirements or needs. In these cases new services will be built, or existing ones will be modified based on the system requirements. The project that executes the application development process is the stakeholder who commissions the development or modifi-cation of the service. For each service to be built or modified, a separate development project is started, each following its own V-model (fig. 2).

3.1. Acceptance of the Services

Development and testing of the individual services completes every stage of the process as described in 1.1 above. Only the acceptance test runs differently. The acceptance of an individual service will be done in three different ways by different acceptors.

The service owner

The owner will perform an acceptance test based on the requirements defined for the functionality, the capacity and the quality that the service must deliver. Testing objectives include - amongst others,- does the service correctly perform the business process and are the interfaces implemented as defined in the service contract.

Operations/hosting

All services will eventually be executed in the production environment. It is of the utmost importance that the ser-vice conforms with the operating requirements laid down in the service provision agreement (SPA): the capacity must be sufficient to fulfill the accumulated demand of all services that will use it (requesters), the service must run well in the operational infrastructure (middleware, hardware etc.), and it may not consume all resources available in that operational environment at the cost of other services. An important part of the acceptance of the service by operations/hosting will consist of testing the service against the non-functional requirements.

The project

The final acceptance of the service is done by the project itself. The newly built and modified services must be ca-pable of delivering the requested functionality together with existing services. The emphasis is placed on the integra-tion of the services into a whole. The service will therefore be tested in combination with the directly communicating service providers and requesters.

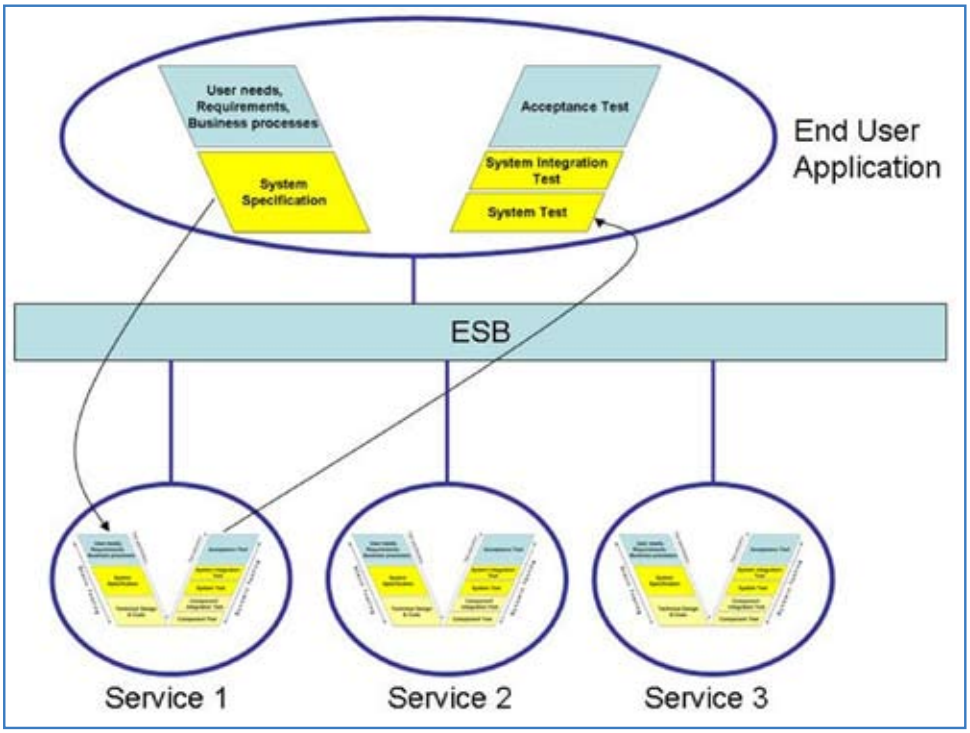


fig. 2 Relation between the development levels

3.2. Acceptance of the Application

The application, as a whole, is accepted by the stakeholders of that application. After execution of a successful service integration test, the process will run as described in 1.1. A system test and a system integration test will be performed. After a successful test of the application, the supplier (internal IT department or external supplier) will hand it over to the accepting party. This party will test the application and the related processes, using the business / end-user requirements and the process definitions as the test basis. During this test the application is tested as a whole and the internal structure of services is not taken into account. The application is accepted if the application and the related processes completely align with the existing processes within the organization.

4. Test Execution

Berlin, Germany

IT Law
Contract Law

German
English
Spanish
French

www.kanzlei-hilterscheid.de
info@kanzlei-hilterscheid.de

k a n z l e i h i l t e r s c h e i d

Test execution is the process of running a test on the component or system under test, producing actual result(s). (ISTQB, 2007). During test execution, starting in a predefined situation, a number of actions are performed, and it is checked whether the results of these actions are consistent with the predicted results that are documented with the test cases. Parts of the system can be tested in isolation or connected to the surrounding systems. This choice de-pends on the goal of the test.

4.1. End-to-end or Isolated Testing

During a system integration or end-to-end test, all communicating (sub-)systems are tested while actually being connected to each other. For instance, think of testing the chain order entry, order processing and finance.

End-to-end testing is, technically spoken, not that difficult if the test is planned for the right moment, i.e. after every part of the chain has successfully passed a system test. However, the difficult part is in the organization that is needed: this involves getting hold of the correct test environments and synchronized databases. That is why we often try to avoid the end-to-end test by proving the correct cooperation a different way. One way of doing this can be by testing the individual interfaces between the various sub-systems.

A SOA based system can be seen as a chain of sub- systems. Therefore we can compare the testing of such a system with end-to-end testing. Here we will also see that, as a result of complexity of environments and test data, we will test the services in isolation where possible. This decision depends on the product risks. In case of a high risk it will be useful to perform an end-to-end test. However, in many cases an isolated test will do.

We can say that, after a service has been modified, it should at least be tested in combination with all the services that communicate with this service. The border of the chain lies at the interfaces that are not changed or affected (see fig. 3). In this figure, S5 has been modified. Therefore a test with all the services in the shaded square is in-evitable.

In this simplified figure we can use the requesters App2 and App3 as input and output devices. In case that these requesters are complex applications, it might be better to use drivers. Stubs are used to replace the services that are called (S11, S12 and S13). Virtualization techniques can be applied to simulate the ever more complex interfaces to legacy back-ends (DB2 of IMS) of external services. More about stubs, drivers and virtualization in 1.4.5 and 1.4.6)

If SOA services are tested in isolation, stubs, drivers and specific test data can be used. The main advantage is that the entire test set (SOA service, stubs, drivers and test data) can be handed over to other parties to be tested.

4.2. Test Environments

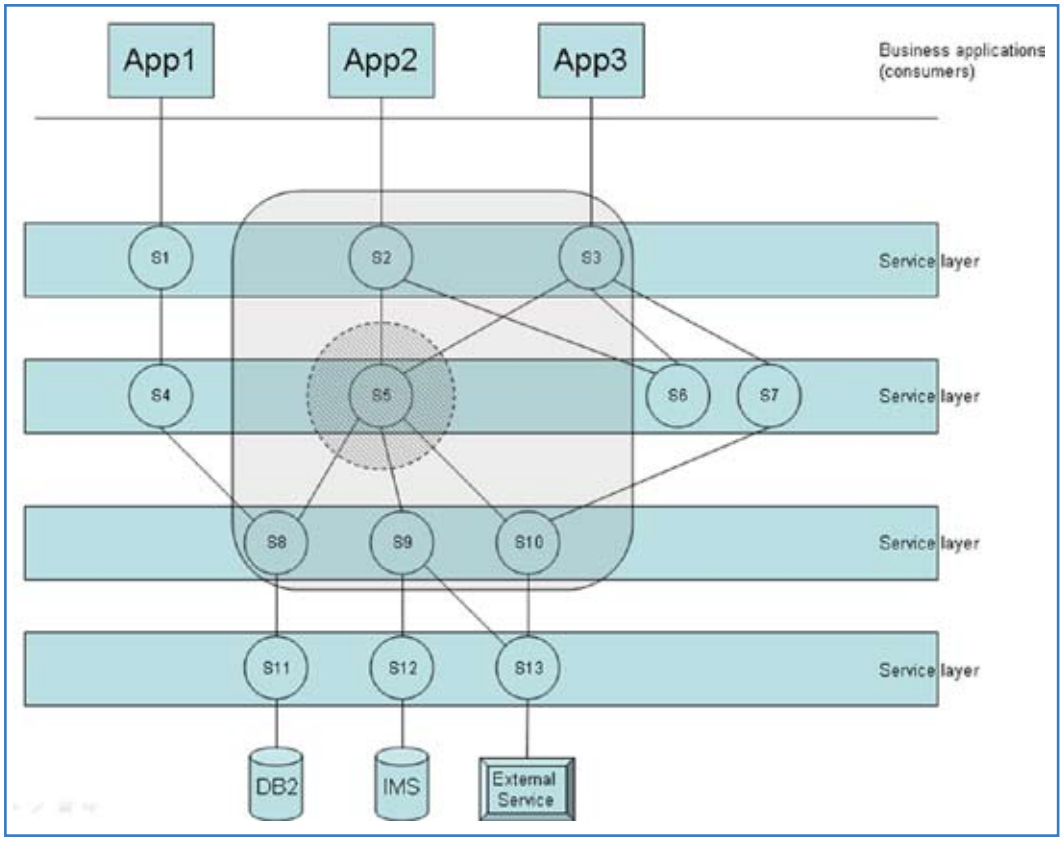
We often see the DTAP configuration of test and production environments. DTAP stands for: Development, Test-ing, Acceptance and Production. These environments are related to the test levels of the V-model. More than one occurrence of each environment can exist simultaneously. The testers should realize that not all tests are executed in all environments. This means, that in the development environment the system will not be connected to the sur-rounding systems, so it will not be possible to perform a system integration test. It is defined in the test strategy which tests will be executed in which test environment.

Responsibility for the test environments

In most cases, the IT department is responsible for the development and test environments and the business for the acceptance and production environments. If and how these environments align depends on the relationship and cooperation between IT and business. In each environment we will find one or more databases. The test environments and databases need to be managed. It is a challenge to have the correct test data in each database and to synchronize the data structures and software changes.

Services in test environments

Testing won't be very difficult if only one service has been changed. The service is tested by the developers in the development environment. System and system integration test are executed in the test environment, in which all services that are needed are available either physically or virtually. It will be something different when services are used that are also used in other projects. In this situation it is important to keep everything consistent and manage the process precisely.



4.3. Test Data

Irrespective of the system development approach or architecture, before we can really start testing, a set of data must be available in the database. Consider parameters like discounts and tax percentages and tables with fixed data. If we do not know the contents of the database, we cannot determine which data to use in test cases and predict the expected results. In contrast to what is often thought, it is not always useful to use production data during the test. It may occur that the data needed for certain tests is not available in the production database. Anyway, in many cases privacy laws prohibit the use of production data.

It is a challenge to keep the test data consistent, especially in complex SOA environments, because various services often use a specific set of data. To

reduce the need for consistency of test data, we can test the services in isolation using synthetic data. Synthetic data means that the data in the database are valid, but that this data have been created by the testers instead of having been retrieved from the production database.

The use of synthetic data has advantages: the database only contains data that the tester needs, not more and not less. Moreover, the tester knows the contents of the database. This is why he can easily predict the results of a test. A database with synthetic data at least contains all data types that are needed for a test.

Synthetic data can also be used during the service integration test. In this case, the data must be consistent in all databases that are used in the test; relation numbers, account numbers, addresses and so on must exist in all databases. Data generators are available that generate consistent data in all databases.

4.4. Maintenance and Management

Management of test environments

Unfortunately, many organizations forget that the test environment must be regarded as the testers' production environment which is very important to them. Compared to the production environment, the test environment is often seen as a secondary item and often lags behind regarding management and maintenance. This may lead to a situation that the test environment is not available when needed, in which case the test projects lag behind in their schedule. The test manager must therefore enter into agreements on the management of the test environments, e.g. in the form of a Service Level Agreement (SLA).

Test data management

Test data can be used by many people, not only by testers, but also by system developers who just want to try out something. When the management process of the test data is organized, clear arrangements must be made on the ownership of the test data. For each test level, the test data will differ because the goals of the test levels are differ-ent. We should pay attention to this when organizing the test data management. It may occur that different versions of the software will need different test sets and different test data. Maintenance and generation of test data can be time-consuming and costly.

4.5. Stubs & Drivers

When a SOA service is tested in isolation, the surrounding services can be replaced by software that simulates the functionality of those services. These pieces of software are known as stubs and drivers. Drivers are services that call the service under test using the correct (test) data. Basically, the drivers replace the service requesters. The service itself too can call other internal or external services. These called services can be replaced by stubs. Stubs react on the request of the requester in a predictable way. System developers deliver stubs and drivers along with the SOA services. This is done at the testers' request, who specify the properties the stubs and drivers must have. The testers formulate the requirements in such a way that they can use them in their predefined test cases, so they are able to predict the results of the test. This creates the possibility to use synthetic data.

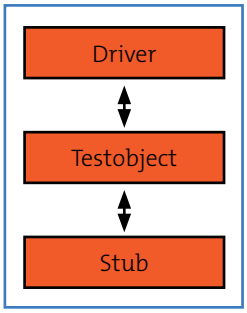


Fig. 4 Relationship between test object, stub and driver

Stubs are not only suitable to replace individual services; it is possible to simulate entire sub-systems. Particularly when interfaces to external service suppliers are used, stubs can serve well; the test environments at the external supplier's site are often outside the range of influence of the testers. Furthermore, there are often additional charges related to the execution of tests with external suppliers. By replacing these external systems with stubs, the tests are easier to manage and costs can be reduced.

4.6. Service virtualization

Tools exist that are capable of replacing and simulating services. This is also known as service virtualization. These tools act as stubs and are able to capture actual data that are supplied by existing services, modify (encrypt) these data and use them to simulate the service. The tester can also use the tool to define business rules, so that the tool will react predictably when a service under tests sends out a request. A throttle mechanism allows the tester to man-age the number of responses per unit of time, which is useful during performance testing.

The virtualization tools give the tester full control of and responsibility for stubs and drivers as well as the data to use. This offers more flexibility during preparation and execution of (regression) tests than using programmed stubs and drivers, and the results of the test are better predictable.

5. Test Management

All test activities described above are managed by a test manager, who is responsible for the complete test process. During project preparation, the test manager defines the test strategy based on the product risks. The test strategy specifies the conditions for system acceptance and the test levels to be performed in the test project. This is the basis of test estimation and planning. The appropriate tests must be planned and performed in order to be able to judge the quality of individual services and the system as a whole. Therefore the test manager too will have re-quirements on the sequence of delivery of the parts of the system, test environments and test data. Often, it is the test manager who is responsible for the integration of the services in the SOA environment.

6. Conclusion

Do we always have to test SOA services in a chain or will isolated testing of these services do?

Testing of SOA based applications, from the end-users point of view, is no different from testing any other applica-tion. From the operations, governance and maintenance perspective, however, there is a big difference. This differ-ence is due to the fact that SOA services are individual pieces of software that perform their own tasks, which are triggered by requests from other services or user interfaces. Besides the impact of these properties on the develop-ment of SOA based systems, the impact on the test process is important. We are often inclined to test the services in a chain as much as possible. However, this is complex, demands for specific test environments and is, because of that, disproportionately expensive. This is why we should test SOA services in isolation whenever possible and only prepare and execute an end-to-end test in case product risks demand it.

Who will accept the SOA based applications, and who accepts the individual services?

By regarding the services as individual programs, each with their own stakeholders and clearly defined tasks and interfaces, they can easily be tested in isolation. This way we can speak of staged acceptance: acceptance of the individual services and acceptance of the application as a whole, in which case the service owner accepts the service and the organization the application.

Who must test when one of the services has been changed?

A SOA service is always used in a system of services. When a service within a system is changed, it impacts the directly connected services, either as requester or as provider. Therefore we will have to perform a test on all these directly communicating requesters and providers. To requesters and providers, for which the change of the service implies a change in functionality, it means that the changed service must be accepted. In all other cases, the requesters and providers must pass a regression test, in order to check

that the change in the service has no impact on the existing functionality and quality.

Wind up

This article explains why testing of SOA based systems requires special attention. Within a system consisting of services, many parties are interdependent; it is therefore important to manage the test process in a structured way, guided by the product risks defined by the stakeholders. It is important to apply a test process that is based on re-quirements and related risks and that gives insight into the consequences of one or more services being changed.

This article has been published before in Dutch in "*Architectuur móet bijdragen*" - Academic Services - LAC 2009

References

Iris Pinkster et al, Successful test management: an integral approach, - Springer Verlag 2004

Standard glossary of terms used in Software Testing Version 2.0 - ISTQB 2007

Biography

Chris C. Schotanus has more than thirty years of experience in IT, more than fifteen of which were spent in the field of testing, test management, setting up test organizations and test automation. As a principal test consultant for Logica in the Netherlands, he advises organizations regarding test policy and test organization, both in the Netherlands and internationally. He is the co-developer of Logica's methodology in this field and co-author of the book '*TestGrip: Gaining Control of IT Quality and Processes through Test Policy and Test Organization*' (2007). He is also one of the founders of TestFrame®, Logica's method of structured testing, and co-author and editor-in-chief of the book '*TestFrame, a method for structured testing (2008)*'. Contributions by him can also be found in Logica's book, '*Successful Test Management: An Integral Approach (2004)*' and several other books and publications on testing. Chris gave a tutorial on SOA and Embedded Software testing on QA&Test 2009. He teaches various testing-related courses and gives regular presentations at different national and international congresses.

The Risk Analysis Tool & Conversation

Improving software quality simply by using Risk Analysis to create conversation.

by Amy Reichert

Every release of software involves elements of risk. Some are minor, but many are major, and several are absolutely critical. With every new release of software, software companies dump dozens of issues in a clients lap, albeit not intentionally. As professional software tester it's an essential part of the job to find critical issues before the software goes to a customer. However, in my professional experience as software tester, I typically work with one hand restricted by deadlines based on business needs, and with the other hand juggling multiple integration points trying to determine which ones to test first, if at all.

How as a tester do you know where to start? My goal is to give you a few straightforward, reasonably quick methods to perform risk analysis to better determine where to focus testing. Specifically, I want to address how to engage your developers in a risk conversation, so that you gain knowledge of where the holes, or bugs, in the software are. In this way, major issues are found sooner rather than later in the development cycle. More importantly, it's a communication path between tester and developer that is critical to achieving a quality software release in a tight timeframe, regardless if your company prescribes to a software testing methodology of any kind, or none at all. Building a positive relationship with developers is crucial to quick, business- paced testing.

The risk analysis conversation identifies critical paths in the code, integration points, and unknowns. It enables you to visually cue on facial expressions and comments, as well as discussions to discover areas in the code that may be ripe for defects. Essentially, you're using different skills – listening, deciphering, contemplation and reading physical signs, like facial expressions, mood, and body language. First, let's review a couple of basic tools you can use to start the conversation, then how to read and interpret the discussion and finally, how to focus your testing efforts on what you've learned.

A Simple Tool

The Risk Analysis data tool is created using whichever type of application you prefer. It can be created as a simple table or as a spreadsheet at the degree of complication that serves your needs. The point is to use it as a tool to enable software testers to prioritize test cases on projects with the aid of development staff and where possible, product and project managers. It's not a lengthy or an overly complex document that no one has the time to read,

Risk Analysis – Project: Patient Insurance Evaluator

Requirement or Defect	Company Risk	Customer Exposure	Error History	Integration Impact	Code Complexity	Modification Frequency	Requirement Weight	Risk Score	Risk
Define insurance carrier for patient	10	10	5	0	2	2	5	34	Medium
Display insurance payment verification	10	10	2	10	10	10	10	62	High
Display insurance payment decision	10	10	5	5	5	5	5	45	High
Print form for patient authorization for insurance payment	5	10	0	0	2	0	5	22	Low
Key:	High => 40								
	Medium => 30								
	Low =< 30								

Figure 1

learn, or process. It's meant to be a relatively quick and painless method to determine where to focus testing.

We'll discuss three sections typically included to gather input from Product Managers, Development Managers, and the QA tester.

The Product Management section contains three areas, where they provide input on the importance of the project to the organization as a whole. They provide input to weight these areas in order to "value" how the project affects the product overall. As a tester, we're interested in getting their "global" view of the software suite. They can provide useful information on the following: Company Risk, Company Exposure, and Error History. The first two are somewhat obvious, what do they believe is the risk to the company if critical defects are found by customers. Additionally, being involved with the customer base gives them information on what portions of the software generate customer complaints. It may be that the project is completely new and has no history of errors, however you can use the history from the general functional area if it's applicable. If not, it can be removed or valued after the application completes either beta or pilot testing. Figure 1 is a sample format created as a table in Microsoft Word.

Each slot is assigned a weight level on a scale from 1 to 10, where 10 is high and 1 is low. It is a subjective evaluation based on the group's experience and knowledge of the company. It's essential to spend time discussing it so that the data created is as accurate as possible. One way to get this communication started is to fill out the weight values yourself as the QA.

Personally, I fill the weight values based on my opinion and send it out to the project group two or three days prior to the meeting. I use my experience with the application, the customer base, and knowledge of the areas in which defects are found. I do this because it facilitates the discussion, because it becomes a debate. It seems easier for many people to debate or argue a value they don't agree with, than come up with a value from scratch. As long as it facilitates a discussion, it works. The object is met, once a group gets going in a discussion, you'll get the testing information you need. It's creating this open communication that provides the value.

In cases where you feel you lack experience to set the values, enter in an estimate. The idea is to get them to review your values and discuss. Remember, the value in the tool is to get the conversation going, as long as you do that, you'll get the information you need.

You can even make the grid simpler than the one used in Figure 1. If you truly don't want to set specific numeric values, try using text:

Requirement or Defect	Company Risk	Customer Exposure	Error History	Integration Impact	Code Complexity	Modification Frequency	Requirement Weight	Risk
Define insurance carrier for patient	High	High	Low	Medium	Medium	Low	High	High
Display insurance payment verification	High	High	High	High	High	Low	High	High
Display insurance payment decision	High	High	Low	Low	Low	Low	Medium	Medium

Figure 2

You still have to set a value, but some groups find it clearer than using numeric values. Either works, just use one that suits your group or modify it to fit your needs.

After the project ends, you can review your values with the actual results and see how accurate they were. Results can vary, as can your access to them. Typically, you can usually find a support or customer service resource for your application and see if they have data or information on where and what defects are reported by customers. Minus those types of resources, you can monitor defect reports coming in and get data as to where in the functionality the defects are found. If your testing group already produces metrics for management reporting, try utilizing those to your advantage.

Now, if you're lucky enough to have a development manager or a lead developer for the project, they are responsible for the next fields: *Integration Impact, Code Complexity, and Modification Frequency*.

Integration impact is the amount the functionality affects other applications within the suite. Granted, each customer can usually pick which applications they use together, so it's not always easily defined. Prior to the conversation with development, find out which applications yours is typically integrated with and how. In this way, you can bring those applications into the conversation to ensure integration is considered during design and coding.

Code complexity is a way to value how difficult the functionality was to code. The more lines of code, or the more complex the code, the higher the likelihood of errors.

Modification Frequency is an estimate of how often they believe the code will change. You need to know how frequently they think the code will change based perhaps on future plans. As a tester, we're not always aware of new functionality that has started the development process. Developers are typically aware of any new additions that have just started or are being discussed.

So, now you're in the meeting. You've entered values that have caused the conversation to start. Now, how can you keep it going? Moving through each requirement, in our previous example is one way to keep the conversation going. It'll be up to you to keep them focused. Bring up your list of integrated applications, and see what response you get. If you see a room full of surprised faces, that's a problem! If you find out they're not aware of other applications – that's another problem.

When I asked a group of my co-workers if they believed the tool was working, I received a very consistent response. Scott Phillips, lead tester for a nursing application responded, "*I think it's important (the tool), but the conversation it promotes is super important.*" When I asked if they felt they got useful feedback from development – over 95% said yes. Sandy Armitstead, professional QA and RN said, "*Yes, there's always a surprise in what they consider a priority versus what I do.*"

In our first pilot project using this simple risk analysis tool, we expected not only indifference but reservation and negativity. In our organization, QA process changes are usually short-lived and often met with complete disregard. However, when we got the group together and showed them the form, they expressed a sigh a relief. It was straightforward and simple to understand.

ISTQB® Certifications, Wherever you Choose

Special Offer for Testing Experience Readers*:
Receive Amazon Coupons + Free ISTQB® Exam Vouchers

SELA Group is a global company with over two decades of track record in IT training and consulting. SELA is recognized as leader in training of high quality IT personnel for the thriving and demanding IT and hi-tech community.

SELA Group – is an Accredited Training Provider of ISTQB® since 2004.

Our Training is an original development of SELA that reflects many years of practical experience.

Our Trainers wish to share their vast experience in conducting training worldwide and from consulting to the hi-tech industry.

CTFL - ISTQB® Certified Tester Foundation Level

CTFLTM - ISTQB® Certified Tester Advanced Level - Test Manager

CTFLTA - ISTQB® Certified Tester Advanced Level - Test Analyst

Course	Country	City	Date
CTFL	Israel	Tel Aviv	April 6,8,13,15
CTFL	India	Pune	April 12-15
CTFL	Singapore	Singapore	April 19-22
CTFL	Canada	Toronto	April 26-28
CTALTA+CTALTM	Canada	Toronto	April 26-30
CTALTA	Singapore	Singapore	April 26-30
CTALTA	Israel	Tel Aviv	May 2-6
CTALTA+CTALTM	Canada	Toronto	May 24-28
CTFL	Canada	Toronto	May 25-27
CTALTM	Israel	Tel Aviv	June 6-10
CTFL	Israel	Tel Aviv	June 7,9,14,15
CTFL	Canada	Toronto	June 28-30

Course	Country	City	Date
CTALTA+CTALTM	Canada	Toronto	June 28-July 2
CTALTA	Israel	Tel Aviv	July 4-8
CTFL	India	Pune	July 5-8
CTFL	Singapore	Singapore	July 12-15
CTALTM	Singapore	Singapore	July 19-23
CTALTM	Israel	Tel Aviv	August 1-5
CTFL	Israel	Tel Aviv	August 2-5
CTALTA	Israel	Tel Aviv	August 15-19
CTALTM	Israel	Tel Aviv	August 22-26
CTFL	Canada	Toronto	September 20-22
CTALTA+CTALTM	Canada	Toronto	September 20-24
CTFL	India	Pune	October 4-7
CTFL	Singapore	Singapore	October 11-14

* Subject to terms and conditions

solutions@sela.co.il
+972-3-6176626
www.selagroup.com

SELA GROUP

ISTQB® Certified Tester

They knew which pieces they needed to specifically respond to, and which to defer to product management. I had filled in the values myself, which caused them to consider what I'd put down as the value and either agree or argue against it. Sometimes a good argument point keeps the conversation valid and moving forward. During the meeting, we also discovered more than one integration point that had not been considered, as well as one instance where developers in group A, for example, were developing a function in complete contradiction to the developers in group B. We had a full-blown code discovery occurring. I learned more in that first meeting about the application I tested than I had in months. I also learned at which integration points they tended not to discuss with other applications.

It honestly amazed me how well the communication worked and how our relationship as a work group has grown more positive and productive. When you promote communication and conversation in this way, it creates less of a competitive atmosphere and more of a collaborative one.

I am still surprised at how well it continues to work despite the tool's simplicity. It's incredible what communication will do for quality.

When I asked co-workers if they believe they could discern the risk based on requirement and design review meetings, the overwhelming response was "no". So, I asked them, is it easier to just discern the risk via conversations with developers in general, no tool needed? Scott Phillips, our lead nursing application tester said, *"I don't think so. I think that would start a slippery slope towards when the conversations didn't occur at all. Use of the tool forces those conversations."*

Despite the simplicity of the concept, it continues to amaze me as a tester how many issues could be resolved or completely avoided with communication. Using the risk analysis tool to generate conversation with development saves testers significant time. It keeps us from running from one place to the next looking for answers, and it allows us to do our job better and add more quality to the product than just testing alone. It allows us to focus our testing based on the risk values and conversation. It also allows us to find out where there are possibly holes in the code or where integration points have been missed. We can improve the quality while we're still early in the coding process and before it ever gets seen by any customers.

All things considered, the Risk Analysis tool provides an effective and efficient method of assessing risk and prioritizing testing by creating a conversation between development and testing. It provides a simple method to target testing when testing deadlines are tight in both time and resources. It opens lines of communication from development and product management to the testing team. Finally, it's a useful, simple tool to assist testers in minimizing the impact to customers when full coverage testing does not occur.

Risk Analysis Methods

You can easily find information by searching the internet and pouring through the reams of varieties of methodologies and complexity levels involving performing risk analysis.
Traditionally, an area controlled by Project Managers, less complex matrices can be used to efficiently analyze risk depending on your need. Software is frequently developed without Project Management input or availability.
For my use, I find simpler is better and more likely to be accepted, considered, and used.

What is Risk?

It is the possibility of a negative, or undesirable effect.
Naturally, we seek to avoid risk. If weather forecasters tell us it's going to snow 37 inches overnight – we're not going to drive to the office, we'll opt to work remote and avoid the dangerous weather and traffic.
In our software world, it's loss or gain of a business reputation based on released software that performs at an exceptional level, or that fails to function properly causing harm to a living being, or a business.
In risk analysis, we focus on what affects could be negative and which could be positive.
Where do the risks in our software reside? We tend to focus on the negative risks.

Biography

Amy Reichert is a seasoned software tester, with 14 years of professional QA experience. She has a BA degree in English from the University of Colorado at Denver. She has a wide variety of QA experience in companies producing ERP, architectural design, e-commerce, and currently works in a leading medical software corporation. Amy holds her testing certification (CTFL) with ASTQB.

An Overview of "A Visual Approach to Risk-Based Integration Testing"

by Neil Pandit

With reduced budgets, resources and a need to deliver projects faster, Risk-Based Testing (RBT) is an area that is increasingly being seen as an approach that stakeholders and project managers are keen to adopt.

RBT is about developing an objective technique to prioritize the testing effort, with a view to reducing the overall testing effort, whilst minimizing any increase in risk exposure to the organisation. It is, therefore, an ideal approach to be applied to projects where there is a combination of limited time, money and available people. Unfortunately, there are often misconceptions about the use and objectives of RBT, which can lead to it being perceived as a "taboo" technique within some organizations.

This short article will provide an overview of the presentation that I delivered at EuroSTAR 2009 entitled "A Visual Approach to Risk-Based Integration Testing." It will highlight some of the misconceptions and problems encountered when using RBT and how we have addressed these successfully at Sopra Group by using an RBT approach to test complex system integration projects.

Misconceptions of RBT

Clarification of Objectives of RBT – The objectives of RBT can often vary in organizations, depending on who you speak to. From a testing perspective, the objective is typically to mitigate risk, from a project manager's perspective it is often to reduce costs but not compromise on quality, whereas from the business areas it can be viewed as a reason by testing to do less testing, and is sometimes therefore perceived as increasing the overall risk.

Independent of software delivery - Another misconception is that the RBT approach can be developed independently of software delivery. A prioritized order of execution can be produced, but if your systems and components are delivered in a different order by your development partner, then you are effectively simply executing scripts based on delivered functionality.

No defects in later test phases - There is a misconception that RBT will find **all** the major defects, so there is great surprise when a high-severity defect is found in UAT. The two issues here are:

1. It is "risk"-based, which implies there will be risks remaining and therefore potential defects.
2. The defect may be a major-severity defect, but this should be in a non-critical system or process. Again, as a result of defects remaining, the perception by the business is that RBT still leaves you with defects.

Problems with RBT

In addition to the above misconceptions, testing of major system integration projects can result in a number of additional problems.

Obtaining "objective" risk assessment of multiple integrated systems - It may be straightforward to apply a RBT approach when testing a single system, as it usually involves a single technical / business resource assessing the risks, but with a large number of systems and interfaces and potentially a number of resources it will be harder to objectively assess the risks of failure of systems.

Focus is on systems and not interfaces – Typically when applying RBT, the focus tends to be on systems, and so interfaces and data transfers can get overlooked. Often, the business users may not be aware that data is being transferred, as they may not be aware of the lower technical architecture of how data is transferred from one system to another.

Reporting does not focus on risks mitigated but on "numbers" of scripts executed - Although execution is risk-based, progress reporting tends to continue to focus on the number of scripts that have been run. Typically, project managers are still looking for how many scripts have been run, rather than the risks that have been mitigated. Progress reporting also lacks visible representation to key stakeholders such that it lacks clarity over where the problem systems or interfaces are within the architecture / process.

Taking into consideration the misconceptions and the problems with large system integration projects, an approach that I have successfully applied uses the concept of heatmaps.

Heatmaps – A Potential Solution

A heatmap is a visual representation of data or information through the use of colors. This method of communication is accessible to all levels of users and is commonly used in other sectors, such as weather forecasts or stocks and share price fluctuations.

At Sopra Group, our approach of applying heatmaps for risk-based testing similarly provides a visual representation of the testing priorities through the use of colors. As the most commonly understood diagram on a project is the system architecture diagram, this is used as the basis for applying the heatmap approach.

Approach

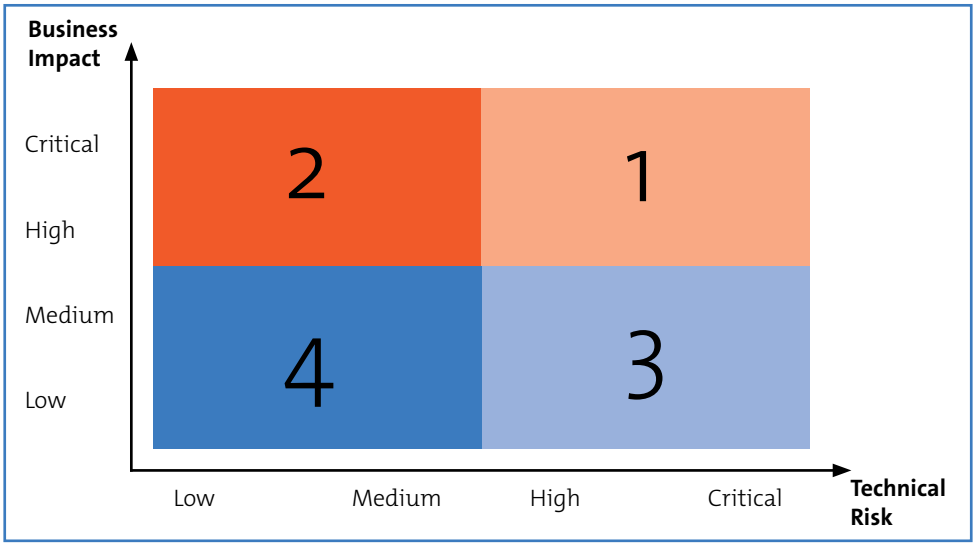
In terms of approach, the first activity is to obtain the system architecture diagram for the system. We then set up a workshop with the technical experts or owners of the systems. During the workshop we look to get technical input and an understanding of the technical complexities and risks of failure of the systems and interfaces, which we can annotate on the diagram. The principle is to apply a heatmap to the system architecture diagram to demonstrate and agree levels of technical likelihood of failure.

This provides a technical heatmap based on 4 levels of priority such that critical relates to a new system or interface being created, unstable existing production system, new technology / supplier etc. A "low" likelihood would be where there were minor / no changes to a system or interface.

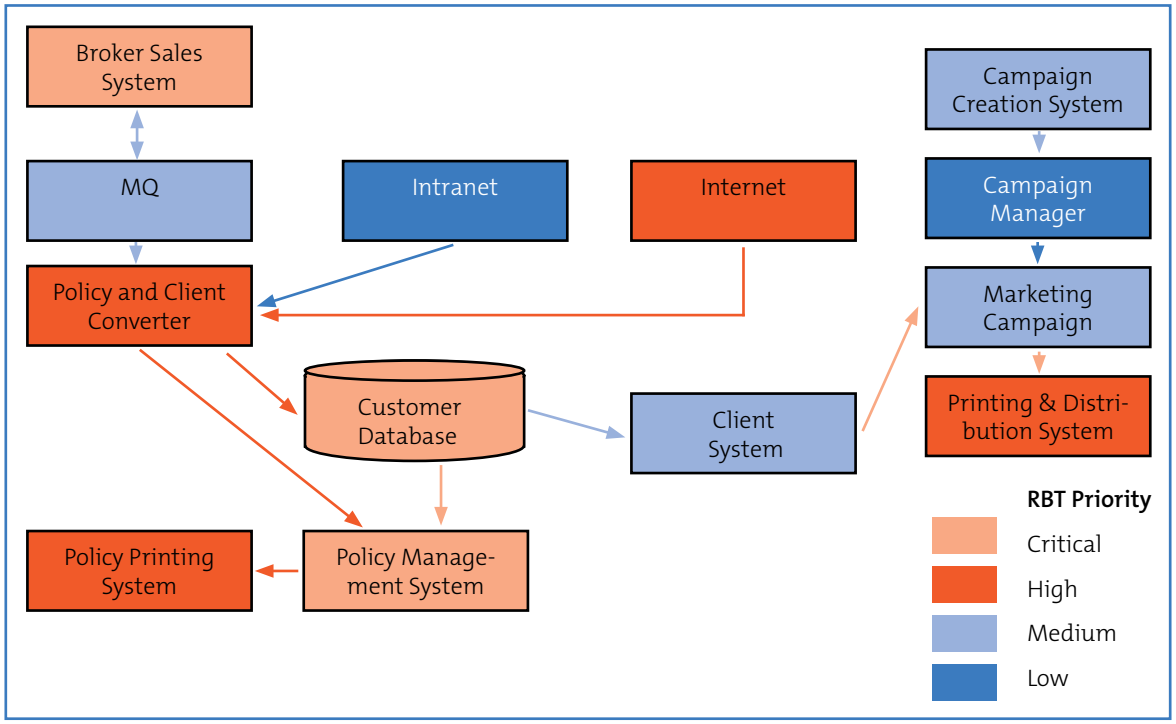
Having created the technical heatmap, we set up another workshop with the business areas to understand firstly the critical business processes and map these to the system architecture diagram. We then perform the same prioritization exercise using 4 levels of prioritization with the remainder of the systems to agree a prioritized business impact of the systems and interfaces.

A critical-priority system may be one where failure incurs a significant regulatory fine, has direct financial / customer impact or reputational impact. A low-severity impact may be a system or interface which may have manual workarounds that can be put in place immediately, or internal systems and interfaces which are non-customer facing.

So, at this point we have a system architecture diagram with the technical likelihood of failure and an architecture diagram with the business impacts of failure of each system and interface. Using a matrix as shown below, we take the business and technical priorities for each system and combine these into an overall prioritized approach. So a system that is considered to be high (orange) from a technical risk perspective and a high (orange) from a business impact perspective will have its overall rating increased to a critical (red) rating.



A simplified example of a system architecture diagram that might be produced is shown below:



As a result of using this technique, we end up with a final combined agreed diagram that not only takes the business impact into consideration but also the technical complexities of the solution.

In terms of benefits of this approach, these are:

- ***Highlights the RBT approach visually*** – The primary benefit is that it enables you to visually prioritize and focus your system integration testing.
- ***Provides ease of understanding for all stakeholders*** - This is a simple method of communicating the RBT approach to all stakeholders, who may not have much knowledge around testing.
- ***Incorporates technical risk into test design*** - Provides testers with a clear focus on test design of scripts ensuring that scripts are created which cover the maximum number of hotspots whilst incorporating the technical risks.
- ***Provides risk-based reporting against specific interfaces & systems*** – Enables you to use the heatmaps to provide visual progress reports on specific interfaces and systems.

Conclusions

This has been a brief introduction to using heatmaps for RBT, but the key points to take away are:

- ***Agree the objectives of RBT*** - RBT will continue to be used within organizations, but will, if not applied correctly, be disregarded by the same people it is intended to benefit (project managers, stakeholders, the business users).
- ***More effective focus on system integration testing effort*** - We need to have more effective focus on the system integration effort, as poorly planned system integration testing and incorrect focus could result in significant time and effort being wasted.
- ***Heatmaps - a possible solution*** – This approach can provide an objective risk- based and visual approach to system integration testing, which combines both the technical risk with the business impact.
- ***Ensures common understanding*** – It provides clarity to stakeholders over what has and what hasn't been tested and can assist with reporting progress visually.

Finally, in terms of the key challenges with this approach, we must remember that the heatmaps and any RBT approach cannot be performed by testing in isolation, and so it is essential to get full buy-in from both business subject matter experts and technical resource. This process is about getting all relevant parties involved in achieving the common objective of agreeing the scope, focus and priority of system integration testing.

Biography

Neil Pandit is currently a Senior Test Consultant at Sopra Group, providing consultancy and thought leadership across all industry sectors. With 14 years IT experience, he has in the last 8 years specialized in test management and consultancy. Starting his career in development, Neil rapidly progressed from managing testing teams to senior test management roles for major financial institutions and system integrators. Neil is currently involved in the practical application of risk-based testing.

Model-based Testing: a new paradigm for manual and automated functional testing

by Dr. Bruno Legeard

Model-based testing (MBT) is an increasingly widely-used technique for automating the generation of tests. There are several reasons for the growing interest in using model-based testing:

- The complexity of software applications continues to increase, and the user's aversion to software defects is greater than ever, so our functional testing has to become more and more effective at detecting bugs;
- The cost and time of testing is already a major proportion of many projects (sometimes exceeding the costs of development), so there is a strong push to investigate methods like MBT that can decrease the overall cost of test by designing as well as executing tests automatically;
- The MBT approach and the associated tools are now mature enough to be applied in many application areas, and empirical evidence is showing that they can give a good ROI.

Model-based testing renews the whole process of functional software testing: from business requirements to the test repository, with manual or automated test execution. It supports the phases of designing and generating tests, documenting the test repository, producing and maintaining the bi-directional traceability matrix between tests and requirements, and accelerating test automation.

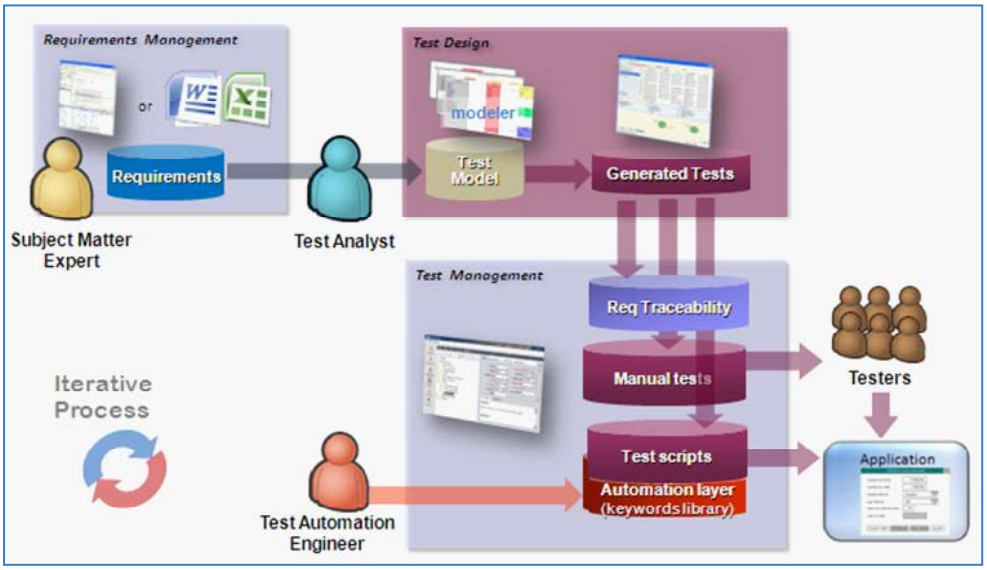


Figure 1. The MBT process

This article addresses these points by giving a realistic overview of model-based testing and its expected benefits. We discuss ***what*** model-based testing is, ***how*** you have to organize your process and your team to use MBT, and ***which*** benefits you may expect from this software testing industrialization approach.

What is MBT?

Model-based testing refers to the processes and techniques for the automatic derivation of abstract test cases from abstract formal models, the generation of concrete tests from abstract tests, and the manual or automated execution of the resulting concrete test cases.

Therefore, the key points of model-based testing are the modeling principles for test generation, the test generation strategies and techniques, and the implementation of abstract tests into concrete, executable tests. A typical deployment of MBT in industry goes through the four stages shown in Figure 1.

1. **Design a test model**. The model, generally called the *test model*, represents the expected behavior of the system under test (SUT). Standard modeling languages such as UML are used to formalize the control points and observation points of the system, the expected dynamic behavior of the system, the business entities associated with the test, and some data for the initial test configuration. Model elements such as transitions or decisions are linked to the requirements, in order to ensure bi-directional traceability between the requirements and the model, and later to the generated test cases. Models must be precise and complete enough to allow automated derivation of tests from these models;
2. **Select some test generation criteria**. There are usually an infinite number of possible tests that could be generated from a model, so the test analyst chooses some test generation criteria to select the highest-priority tests, or to ensure good coverage of the system behaviors. One common kind of test generation criteria is based on structural model coverage, using well known test design strategies such as equivalence partitioning, cause-effect testing, pairwise testing, process cycle coverage, or boundary value analysis (see [1] for more details on these strategies). Another useful kind of test generation criteria ensures that the generated test cases cover all the requirements, possibly with more tests generated for requirements that have a higher level of risk. In this way, model-based testing can be used to implement a requirement and risk-based testing approach. For example, for a non-critical application, the test analyst may choose to generate just one test for each of the nominal behaviors in the model and each of the main error cases; but for one of the more critical requirements, she/he could apply more demanding coverage criteria such as process cycle testing, to ensure that the businesses processes associated with that part of the test model are more thoroughly tested;
3. **Generate the tests**. This is a fully automated process that generates the required number of (abstract) test cases from the test model. Each generated test case is typically a sequence of high-level SUT actions, with input parameters and expected output values for each action. These generated test sequences are similar to the high-level test sequences that would be designed manually in action-word testing [2]. They are easily understood by humans and are complete enough to be directly executed on the SUT by a manual tester. The test model allows computing the expected results and the input parameters. Data tables may be used to link some abstract value from the model with some concrete test value. To make them executable using a test automation tool, a further concretization phase automatically translates each abstract test case into a concrete (executable) script [3], using a user-defined mapping from abstract data values to concrete SUT values, and a mapping from abstract operations into SUT GUI actions or API calls. For example, if the test execution is via the GUI of the SUT, then the action words

are linked to the graphical object map using a test robot. If the test execution of the SUT is API-based, then the action words need to be implemented on this API. This can be a direct mapping or a more complex automation layer. The expected results part of each abstract test case is translated into Oracle code that will check the SUT outputs and decide on a test pass/fail verdict. The tests generated from the test model may be structured into multiple test suites, and published into standard test management tools such as HP Quality Center, IBM Rational Quality Manager, or the open-source tool TestLink. Maintenance of the test repository is done by updating the test model, then automatically regenerating and republishing the test suites into the test management tool;

4. **Execute the tests**. The generated concrete tests are typically executed either manually or within a standard automated test execution environment. Either way, the result is that the tests are executed on the SUT, and we find that some tests pass and some tests fail. The failing tests indicate a discrepancy between the SUT and the expected results designed in the test model, which then needs to be investigated to decide whether the failure is caused by a bug in the SUT, or by an error in the model and/or the requirements. Experience shows that model-based testing is good at finding SUT errors, but is also highly effective at exposing requirements errors [1], even way before executing a single test (thanks to the modeling phase).

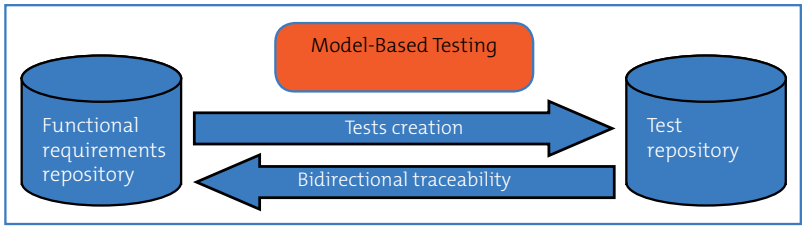


Figure 2. Relationship between both repositories (tests and requirements).

Requirements traceability

The automation of bidirectional traceability between requirements and test cases is a key aspect of the added value of MBT. *Bidirectional traceability* is the ability to trace links between two parts of the software development process with respect to each other. The starting point of the MBT process is, as usual, the informal functional requirements, use cases, descriptions of business processes and all other factors that provide the functional description of the application being tested. To be effective, requirements traceability implies that the requirements repository should be structured enough so that each individual requirement can be uniquely identified. It is desirable to link these informal requirements to the generated tests, and to link each generated test to the requirements that it tests.

A best practice in MBT, supported by most of the tools on the market, consists in linking model elements such as decision points and transitions to the relevant requirements. From these links in the test model, test generation tools ensure the automatic generation and maintenance of the traceability matrix between requirements and test cases.

Test repository and test management tools

The purpose of generating tests from the test model is to produce the test repository. This test repository is typically managed by a test management tool. The goal of such a tool is to help organize and execute test suites (groups of test cases), both for manual and automated tests.

In the MBT process, the test repository documentation is fully managed by automated generation (from the test model): documentation of the test design steps, requirements traceability links, test scripts and associated documentation are automatically provided for each test case. Therefore, the maintenance of the test repository needs to be done in the test model.

Roles in the MBT process

The MBT process involves three main roles (see Figure 3).

1. The **Test Analyst** interacts with the customers and subject matter experts regarding the requirements to be covered, and then develops the test model. He/she then uses the test generation tool to automatically generate tests and produce a repository of test suites that will satisfy the project test objectives.
2. The **Subject Matter Expert** is the reference person for the SUT requirements and business needs, and communicates with the test analyst to clarify the specifications and testing needs.
3. The **Test Engineer** may be a tester for manual test execution or a test automation engineer, who is responsible for connecting the generated tests to the system under test so that the tests can be executed automatically. The input for the test engineer is the test repository generated automatically by the test analyst from the test model.

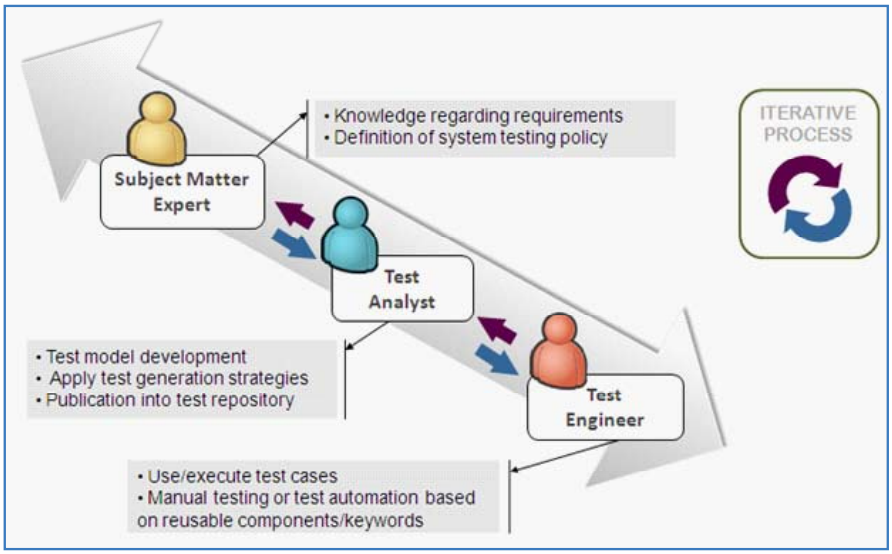


Figure 3. Main roles in the MBT process.

The test analyst is responsible of the quality of the test repository in terms of coverage of the requirements and fault detection capability. So the quality of his/her interaction with the subject matter expert is crucial. In the other direction, the test analyst interacts with the test engineer to facilitate manual test execution or test automation (implementation of key-words). This interaction process is highly iterative.

Testing nature and levels

MBT is mainly used for functional black-box testing. This is a kind of back-to-back testing approach, where the SUT is tested against the test model, and any differences in behavior are reported as test failures. The model formalizes the functional requirements, representing the expected behavior at a given level of abstraction.

Models can also be used for encoding non-functional requirements such as performance or ergonomics, but this is currently

a subject of research in the MBT area. However, security requirements can typically be tested using standard MBT techniques for functional behavior.

Regarding the testing level, the current mainstream focus of MBT practice is system testing and acceptance testing, rather than unit or module testing. Integration testing is considered at the level of integration of subsystems. In the case of a large chain of systems, MBT may address test generation of detailed test suites for each sub-system, and manage end-to-end testing for the whole chain.

Key factors for success when deploying MBT

In the following, we describe the keys to success when deploying a MBT approach and tool. The key factors for effective use of MBT are the choice of MBT methods to be used, the organization of the team, the qualification of the people involved, and a mature tool chain. Given these factors, you may obtain the significant benefits that we discuss at the end of this section.

1. **Requirements and risks-based method**: MBT is built on top of current best practices in functional software testing. It is important that the SUT requirements are clearly defined, so that the test model can be designed from those requirements, and the product risks should be well understood, so that they can be used to drive the MBT test generation.
2. **Organization of the test team**: MBT is a vector for testing industrialization, to improve effectiveness and productivity. This means that the roles (for example between the test analyst who designs the test model, and the test automation engineer who implements the adaptation layer) are reinforced.
3. **Team qualification - test team professionalism**: The qualification of the test team is an important pre-requisite. The test analysts and the test automation engineers and testers should be professional, and have had appropriate training in MBT techniques, processes and tools.
4. **The MBT tool chain**: This professional efficient testing team should use an integrated tool chain, including a MBT test generator integrated with the test management environment and the test automation tool.

Expected benefits

Model-based Testing is an innovative and high-value approach compared to more conventional functional testing approaches. The main expected benefits of MBT may be summarized as follows:

- Contribution to the quality of functional requirements:
  - Modeling for test generation is a powerful means for the detection of "holes" in the specification (undefined or ambiguous behavior) ;
  - The test phase may start earlier and find more flaws in the requirements repository than a "manual" test design approach.
- Contribution to test generation and testing coverage:
  - Automated generation of test cases;
  - Systematic coverage of functional behavior;
  - Automated generation and maintenance of the requirement coverage matrix;
  - Continuity of methodology (from requirements analysis to test generation).
- Contribution to test automation:
  - Definition of action words (UML model operations) used in different scripts;
  - Test script generation;
  - Generation of the patterns for automation function library;
  - Independence from the test execution robot.

Conclusion

The idea of model-based testing is to use an explicit abstract model of a SUT and its environment to automatically derive tests for the SUT: the behavior of the model of the SUT is interpreted as the intended behavior of the SUT. The technology of automated model-based test case generation has matured to the point where the large-scale deployments of this technology are becoming commonplace. The prerequisites for success, such as qualification of the test team, integrated tool chain availability and methods, are now identified, and a wide range of commercial and open-source tools are available.

Although MBT will not solve all testing problems, it is an important and useful technique, which brings significant progress over the state of the practice for functional software testing effectiveness, increasing productivity and improving functional coverage.

References

[1] M. Utting, B. Legeard, *Practical Model-Based Testing: A Tools Approach*, Morgan Kauffman, 2007.

[2] Hung Q. Nguyen, Michael Hackett, and Brent K. Whitlock, *Global Software Test Automation: A Discussion of Software Testing for Executives*, Happy About, 2006.

[3] Elfriede Dustin, Thom Garrett, and Bernie Gauf, *Implementing Automated Software Testing: How to Save Time and Lower Costs While Raising Quality*, Addison-Wesley Professional, 2009).

Biography

Dr. Bruno Legeard is Chief Technology Officer of Smartesting, a company dedicated to model-based testing technologies and Professor of Software Engineering at the University of Franche-Comté (France).
In 2007, Bruno Legeard authored with Dr. Mark Utting the first industry-oriented book on model-based testing, "*Practical Model-Based Testing: A Tools Approach*", Morgan & Kaufmann Publisher. Bruno is a member of the program committees of several major software testing conferences, such as ICST – International Conference on Software Testing, Iqnite (Germany) and MVV – Model-based Validation and Verification.
He started working on model-based testing in the mid 1990's and has extensive experience in applying model-based testing to large information systems, e-transaction applications and embedded software.

Certified Tester Training

in

Spain

Argentina

Mexico

Chile

info@certified-tester.es

Guest Column

The Importance of Software Configuration Management in Testing

by Nadia Soledad Cavalleri

About software configuration management

There are many definitions of Software Configuration Management (SCM), but I will take one given by Roger Pressman in his book called "Software Engineering: A Practitioner's Approach". He says that software configuration management is a "set of activities designed to control change by identifying the work products that are likely to change, establishing relationships among them, defining mechanisms for managing different versions of these work products, controlling the changes imposed, and auditing and reporting on the changes made."

If someone mentions software configuration management, we will surely think of a development team and environment which continues to apply software configuration management practices. Probably this is because software configuration management emerged as a discipline during the 70s, and was focused in programming without covering other aspects of the software development process. I will not question the fact that we thought of development before testing, but I would like to show the importance of software configuration management in test teams and environments.

Software configuration management in testing

Test teams consist of test managers, test analysts, test designers, scripters, testers and some other contributors. Test environments are formed by configurations and artifacts. Configurations consist of aspects of hardware, software, data bases, operating systems, networking, virtualization, privileges and permissions. Finally, there are test artifacts. Among other test artifacts, we can mention test plans, test cases, test ideas, test scripts, logs, issues and reports.

All of them are likely to change, because they are strongly linked to requirements, use cases, builds, and many other artifacts from other disciplines, such as analyses or development, which are constantly changing. For example, the test plan directly relates to the main project plan, test cases may depend on requirements or use case specifications, test scripts and the logs of their execution are for a specific version of a build, and issues are reported on a build version and probably solved on another different version.

In spite of the fact that the configuration seems to be the more static part, it involves changes such as installation of fixes, migration towards new versions of the software, adding users, assigning roles, and other issues. All of these are some of the situations where change tracking is necessary in testing.

Software configuration management, however, is not only about change management with a version control system. When we talk about software configuration management, we also refer to work being performed in parallel, restoration of the configuration, traceability between artifacts and so on. All of these concepts are also used in testing.

We usually work concurrently, which means that while a member is designing test cases another member may be executing them, or perhaps we have two or more scripters who are developing test scripts at the same time. In this way, scripters share the source code of the automated tests, just as programmers share the source code of the application they are developing.

The restoration of the configuration may be necessary if we have to reproduce a bug in the same version of the application and with the same configuration that it had, when the bug was found.

As I mentioned previously, testing products are associated with products generated by other disciplines. In the midst of changes, traceability is extremely important, because it lets us maintain the integrity of models, data and documentation. In addition traceability helps us to uniquely identify the test product related with a specific version of any other product.

Conclusion

The main reason why software configuration management is important for testing is change, but it is not the only one.

If we recognize the importance of software configuration management in development teams and environments, we must give equal weight to the testing teams and environments because they work in a similar way. Finally, even though both teams work with different artifacts, they have to manage issues of similar complexities.

Nadia Soledad Cavalleri lives in Buenos Aires, Argentina.
Nadia completed her degree in Information Systems Engineering at Universidad Tecnológica Nacional in 2008 and she is currently studying Psychology at Universidad Argentina de la Empresa. In 2009, she obtained the certification IBM Certified Solution Designer - Rational Functional Tester for Java Scripting with a grade of 100/100.
Nadia worked as instructor, web developer, technical analyst, tester, test analyst, test designer and test leader. She worked for projects in several industries mainly in financial, government and telecommunications. She also gave courses in different types of establishments such us schools, universities and government institutions.
Nadia is currently working as test leader in consulting projects of software configuration management.

Test Automation in Various Technologies and Environments

by Saurabh Chharia

After a certain time, the repetitive manual testing of test scripts becomes very monotonous and mundane. Also, for certain types of testing like regression testing, it always helps if somebody or something could continue testing the same thing over and over again. With the current trend of software release cycles becoming shorter and the list of tests in the regression test suite becoming longer, it becomes improbable to do complete manual regression testing with every release. Incomplete and inefficient testing may cause new bugs (defects introduced while fixing a bug), which may go undetected before the release. By automating the process (regression testing in this case), the time needed to test run the full suite is reduced, thereby providing the tester with more time to focus on manual testing of new code and on areas of higher risk.

The concept of automation can be used in all such cases. In short, it's mainly about

- Using software tools to execute tests without manual intervention
- Use of other software tools to control the execution of tests, the comparison of actual with the expected outcome, and other test control & test reporting functions.

Test automation as such refers to automating the manual testing process. Automation can be applied to

- Web applications
- Client-server environments.

The process basically consists of test cases that contain expected results based on business requirements, and also an exclusive test environments with a test database in which the test cases can be repeated whenever the application has been changed.

1. Why Test Automation?

The following points list the reasons behind the need for automation (selective and not exhaustive):

- Running the existing tests on a new version(s) of a program/application
- More tests can be run in a shorter time
- Finding bugs that can only be uncovered with long runs
- Better use of resources
- Testers can do better jobs, machines can run all the time
- Consistency, reliability and repeatability of the tests
- The same tests can be repeated exactly in exactly the same manner every time when the automated scripts are executed, thus eliminating manual testing errors

2. Test Automation in various technology environments

In the following section, we diverge into the various automation possibilities for different environments. Most of the examples (if used) would be explained using QTP as basis.

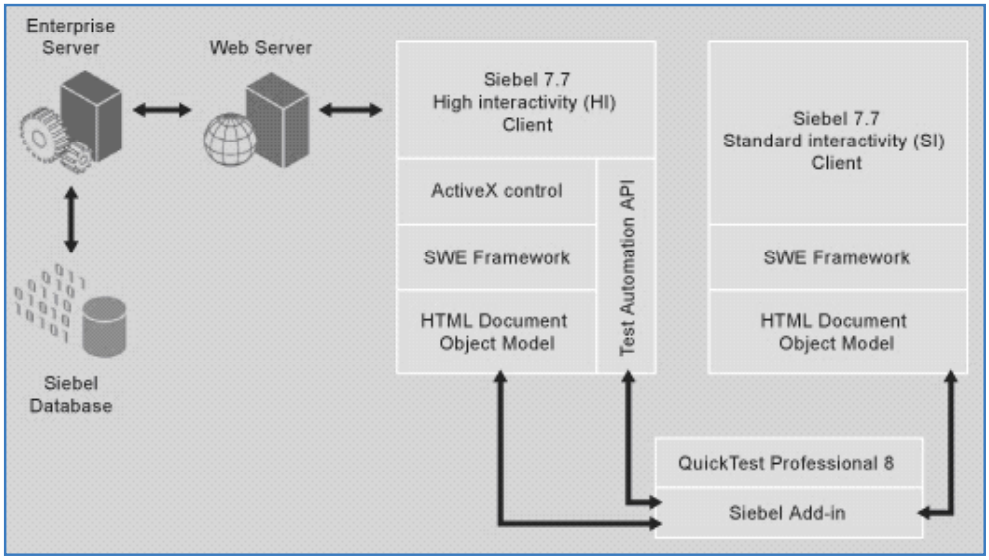
2.1. Siebel

Siebel has two interfaces the High Interactivity client (HI) and the Standard Interactivity client (SI). To automate the HI client, Siebel provides the STA (Siebel Test Automation) by which the automation tools can connect and retrieve information about the controls. For the SI clients, the automation tools need to connect to the HTML DOM to get the required information, as they would for any standard web application.

Integrating with QTP

Integration Architecture

For the Siebel 7.7 High Interactivity (HI) front-end, QuickTest Professional connects to the test automation API to get the properties for all HI controls in the Siebel front-end. To get the information



for all Standard Interactivity (SI) controls inside the HI front-end, QuickTest Professional connects in parallel to the HTML DOM (Document Object Model). By connecting to both levels, QuickTest Professional is enabled to fully support Siebel 7.7 front-ends. For Siebel applications prior to 7.7 (7.0.x and 7.5.x), plain SI front-end QuickTest Professional connects directly to the HTML DOM.

Customization and configuration

QuickTest Professional supports testing on both standard-interactivity and high-interactivity Siebel business applications.

- Standard-interactivity applications download data as and when it is necessary.

This interface is designed for users accessing the application from outside the corporate network.

- High-interactivity applications download the majority of the required data at one time, requiring less navigation. This interface is designed for heavy use and is generally used by call centers.

To enable the Siebel Test Automation interface, the following settings are required:

- Open the configuration file of the Siebel Application to be tested and edit the [SWE] section in this .cfg file by adding the following two entries

-- EnableAutomation = TRUE

-- AllowAnonUsers = TRUE

- When starting this Siebel Application from the browser or from QuickTest Professional, use the following syntax:

http://<hostname>/<application name>/start.swe?SWECmd=AutoOn

To allow secure access to test automation, in which the SWE requires a password, you must also perform the following steps:

- Define a password on your Siebel server. To define a password for test automation on the server, open the .CFG file for the applications you are testing and set the AutoToken switch in the [SWE] section to the test automation password.
    - EnableAutomation = TRUE
    - AllowAnonUsers = TRUE
    - AutoToken = mYtestPassword

The password is case-sensitive and can only contain alphanumeric characters. After you have updated the .CFG file, you must restart the Siebel server. This prepares the server for secure access to test automation.

Send the password in the URL. When starting this Siebel application from the browser or from QuickTestProfessional, use the following syntax http://<hostname>/<application_name>/start.swe?SWECmd=AutoOn&AutoToken=mYtestPassword

If a password is defined in the .CFG file and you do not indicate the password in the URL, the SWE does not generate test automation information.

2.2. SAP

HP Quick Test Professional supports functional testing for enterprise environments like SAP applications with SAP add-in. We can write the scripts for SAP-based applications as it is more user friendly to script in QTPs IDE. eCATT can test only in SAP GUI and has limited error handling and checkpoint capabilities. To test the SAP application from a browser (e.g., net weaver portal application) and to add more flow controls, an external tool like QTP is required.

QTP provides cross-browser testing for SAP applications by supporting all frameworks. It facilitates SAP object-based testing independent of the field positions in the screen layouts.

If an organization really needs automated testing, then timely QA testing of SAP applications along with legacy systems will ensure that all systems work together reliably. The key to building a robust solution for automated regression testing is selecting the right test tool automation framework, and this can be achieved using QTP.

2.3. .NET

There is a wide range of test tools available for test automation for .NET applications, like QTP, RFT.

Rational® Functional Tester supports custom controls through proxy SDK in .NET. The tool includes Script Assure™ technology [Script Assure™ from IBM Rational® software is a set of advanced capabilities for functional and regression testing of Web applications. It significantly reduces the time and effort testers spend on script maintenance tasks and enables earlier testing].

Rational Functional Tester 7.0 supports recording and playback of user applications built using the Microsoft® .NET™ Framework 1.0 or 1.1 and Microsoft® Win32® controls. It also supports testing these controls in a Microsoft .NET Framework 2.0 platform, including recording and testing applications that contain Data Grid View and Masked Textbox controls.

2.4. Web

Web applications are aimed at simplifying the applications. However, automation of the same present us with a greater challenge. The architecture of the web applications make them complex for automation. Furthermore, thousands of components, including custom and third party, make it difficult for recognizing the controls.

The challenge is due to the way the web application runs on the client desktop. It completely runs inside the browser's thread, and hence there is no way to get the information without understanding how the browser dissipates the information. Further, there is no standard to the way the control information is available from the browsers. Hence the automation of web applications is dependent on:

- The web controls, eg. Java, HTML, Dynamic, Flash
- The web browser itself, eg. Internet Explorer, Netscape etc.

Hence, the automation tools need to keep pace with with design changes to the browsers, if they are to provide automation via these browsers.

Let's look into the architecture of the typical Internet Explorer architecture:

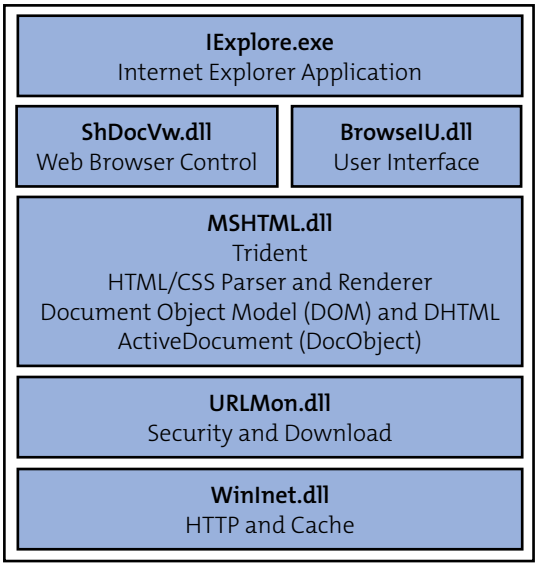
Essential to the browser's architecture is the use of the Component Object Model (COM), which governs the interaction of all of its components and enables component reuse and extensibility. The following diagram illustrates Internet Explorer's major components.

A description of each of these six components follows:

IExplore.exe is at the top level, and is the Internet Explorer executable. It is a small application that relies on the other main components of Internet Explorer to do the work of rendering, navigation, protocol implementation, and so on.

Browsui.dll provides the user interface to Internet Explorer. Often referred to as the "chrome," this DLL includes the Internet Explorer address bar, status bar, menus, and so on.

Shdocvw.dll provides functionality such as navigation and history, and is commonly referred to as the WebBrowser control. This DLL exposes ActiveX Control interfaces, enabling you to easily host the DLL in a Microsoft Windows application using frameworks such as Microsoft Visual Basic, Microsoft Foundation Classes (MFC), Active Template Library (ATL), or Microsoft .NET Windows Forms.



When your application hosts the WebBrowser control, it obtains all the functionality of Internet Explorer except for the user interface provided by Browseui.dll. This means that you will need to provide your own implementations of toolbars and menus.

Mshtml.dll is at the heart of Internet Explorer and takes care of its HTML and Cascading Style Sheets (CSS) parsing and rendering functionality. Mshtml.dll is sometimes referred to by its code name, "Trident". Mshtml.dll exposes interfaces that enable you to host it as an active document. Other applications such as Microsoft Word, Microsoft Excel, Microsoft Visio, and many non-Microsoft applications also expose active document interfaces, so they can be hosted by shdocvw.dll. For example, when a user browses from an HTML page to a Word document, mshtml.dll is swapped out for the DLL provided by Word, which then renders that document type. Mshtml.dll may be called upon to host other components depending on the HTML document's content, such as scripting engines (for example, Microsoft JScript or Visual Basic Scripting Edition (VBScript)), ActiveX controls, XML data, and so on.

Urlmon.dll offers functionality for MIME handling and code download.

WinInet.dll is the Windows Internet Protocol handler. It implements the HTTP and File Transfer Protocol (FTP) protocols along with cache management.

From the above description it's clear that the MSHTML.DLL handles the display and the interaction of the HTML Controls in the Internet Explorer. Once a hook to the instance of Internet Explorer is obtained, using the DOM model we can try to identify the controls and their properties.

A sample application (with code) can be found in the MSDN magazine:

http://msdn.microsoft.com/en-us/magazine/cc163723.aspx

Special browser-based application(s):

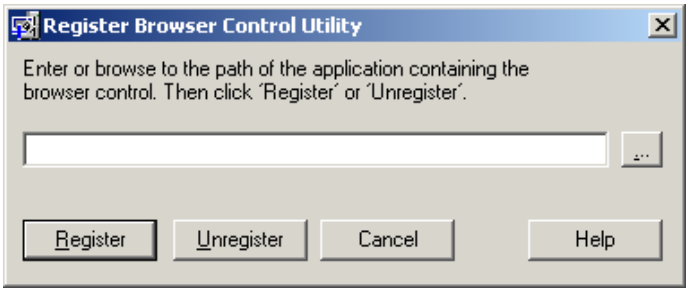
Many of the latest applications are developed with a blend of windows and web controls. The web controls have the capability to show web pages, etc. similar to a web browser. QuickTest Professional supports web controls only running in the known web browsers by default. However, new applications can be registered by using the "Register Browser Controls", which is present under the tools menu of program items.

Registering browser controls

A browser control adds navigation, document viewing, data download, and other browser functionality to a non-Web application. This enables the user to browse the Internet as well as local and network folders from within the application.

QuickTest Professional cannot automatically recognize the objects that provide browser functionality in your non-Web application as Web objects. For QuickTest to record or run on these objects, the application hosting the browser control must be registered.

Note: You can register applications developed in different environments, such as those written in Java, .NET, and so forth.



You use the Register Browser Control utility to define the path of your Web application hosting the browser control. After registration, QuickTest will recognize Web objects in your application when recording or running tests.

Enter the absolute path to the .exe file of the application hosting the browser control, and click on Register. To remove a registered application, enter the absolute path and click Unregister.

After you register an application hosting a browser control using this utility, you must restart QuickTest Professional before you test your application.

Or you can directly modify the registry

1) in the QTP Bin folder you should be able to see a file called mic.ini. Please add the following two lines into it at the end.

```
[ie_hook]
<appname.exe> = yes
```

2) Make the following registry changes:

[HKEY_Current_User\\Software\MercuryInteractive\QuickTest professional\MicTest\Hooks\<appname.exe>]

```
"bbHook" = dword:00000001
"GetMessageHook" = dword:00000001
"WndProcHook" = dword:00000001
```

Test automation is the use of software to control the execution of tests, the comparison of actual outcomes to expected outcomes, the setting up of test preconditions, and other test control and test reporting functions. Commonly, test automation involves automating a manual process already in place that uses a formalized testing process.

In the recent past, a lot of tools have been developed to assist in automatic execution of tests on GUI application. Many of these tools come with the option of recording and playing back the actions that are performed on the Application under Test (AUT). However, if we look at these tools in a little more detail, we will quickly come up with the following questions:

- How does the tool remember which window, which control to act on?
- How does it remember what action has to be performed?

Though the word 'Record & Playback' is a generic "catch-all" to these questions, however let's try to understand how the tool remembers / identifies these objects first.

The following are the possibilities for a test automation tool to identify the objects:

Graphical comparison: This is possible by taking a snapshot of the control on which action is being performed. This way the tool

remembers which object the action was performed on. During play back, the tool then compares the controls with the picture of the control and finds the best possible match and performs the action. In case of a conflict, it compares it with the position in the window to find the required control.

Some of the market standard tools that use this technique for test automation are:

- Eggplant, Test Quest

Contextual comparison: This type of test automation tool remembers the control index from either the top or bottom of the window. Hence 'Second from top' is a way of specifying the control. This kind of tool generally works like scripting tools, without having a recording option.

Some market standard tools that use this technique are:

- Auto-IT, Tomato, Perl-GUITestLibrary.

Object identification: This type of test automation tool identifies the objects in isolation, which is further extended to look for a parent-child mechanism to uniquely identify the objects.

Objects are primarily identified by the properties of the objects, like text, color, height, length, breadth, enabled, disabled etc. These properties are specific to the control class as defined by the development technology. Example: for a text box created in Visual Basic, all the properties as defined by the Visual Studio IDE would be available for verification. Hence the properties exposed by the objects vary based on the technology upon which the object is built. Therefore a text box in a .NET application and a text box in a Java application need not necessarily have the exact same properties.

The tools based on this recognition technique must identify the application technology and based upon that try to access the object properties.

Some of the market standard tools that use this technique are:

- Win Runner, Quick Test Professional, Silk Test, IBM Rational functional Tester

2.5. Windows

In Windows, the user actions are communicated to the application by the user32.dll library. The user32.dll captures all user interactions and passes them on to the application as window messages.

Example: A click on a Windows button is passed on to the button as,WM_LBUTTONDOWN.

How does it do this?

First it tries to find out the control under which the event has occurred. This is done by using the function "WindowFromPoint" function, which is inside its library to get the control present at the required co-ordinates.

It then passes the message (click, double-click, drag, drop, etc). to the control. It's the control's responsibility to do whatever with the message. For example, if the control has been disabled, the message is just ignored, unless it gets a message to make itself "enabled".

However, for the purpose of automation, we need to produce the event on the control. This can be done in two ways:

Interface with the input device (keyboard, mouse, etc) and emulate key press, mouse move etc... (analog automation).

Use the User32.dll and send messages to various controls (context-driven).

2.6. Linux

Linux, a popular variant of UNIX, runs on almost 25 architectures including Alphs/VAX, Intel, PowerPC etc. Shell which is a part of UNIX already provides a lot of powerful commands, which can be used for automating the testing efforts.

Shell, which is the 'command interpreter' for UNIX systems, resides at the base of most user-level UNIX programs. All the commands invoked by us are interpreted by Shell, and it loads the necessary programs into the memory. Thus being a default command interpreter on UNIX makes Shell a preferred choice to interact with programs and write "glue" code for test scripts.

Advantages of using Shell for test automation on Unix/Linux

Some of the advantages of using Shell for test automation on Unix/Linux are as follows:

1. Powerful: Bash provides plenty of programming constructs to develop scripts having simple or medium complexity
2. Extensible: We can extend the Shell scripts by using additional useful commands/programs to extend the functionality of the scripts
3. We can write Shell scripts using default editors available (vi, emacs etc) and can run and test it. No specialized tool is needed for this.
4. We can even generate color-highlighted reports of test case execution, which is of great help.

5. Conclusion

In conclusion, this article demonstrates that Test Automation is achievable in various environments. Hence it reflects the extensibility of Test Automation. Certain key points and settings mentioned in this document provide the Tester to make the AUT compatible to the tool.

Biography

My name is Saurabh Chharia. I have joined 2 months back from Infosys Technologies, Bangalore. I have been into test automation for approx. 3 years and have worked with various automation tools such as QTP, Win Runner, TOSCA and Jameleon.

I was born in Delhi. My father is a C.A, Director in a Marketing Firm and my mother is a housewife. My hobbies include listening to music and traveling. Luckily, I have done my studies and am now pursuing my career in Noida. I come from a school of values (Infosys) and give high regard to this at work as well. I firmly believe that in any business domain or technology, if the quality of the deliverable is appropriate and the process of the organization is streamlined, the product will always be sellable and the customer will be retained.

Cloud QA

by Rodrigo Guzmán

The e-commerce business operates in the complex world of the internet and should be available 24h every day of the year.

The different platforms that clients use to connect to the Internet, browse the e-commerce sites, and make transactions are part of that world. Ensuring usability of the site and good user experience on all these platforms are critical goals for this business.

The role of the Quality Control Team is to ensure a positive user experience and continuous flow of transactions. This is achieved by ensuring that the software behaves correctly under the complex world of the Internet. This work is faced with constant change, where time to market is a priority, resources are scarce and uncertainty is high regarding the use of the platform by users.

Challenges

The Quality Control Team faces three major challenges in this context:

Platforms. Understand and know exactly which platforms the users use to connect to the Internet and browse the site. We need to identify the possible configurations that determine a representative sample for testing. If we do not use that platform, we can`t test the site under real conditions.

Hardware. Have the necessary hardware to configure multiple platforms and create a test lab.

Enlarge the test lab without investing heavily in hardware. Assure autonomy for the quality control team to manage hardware resources and configuration settings. Remove the bottleneck that occurs when we depend on the area of technology.

Test. Build all possible test cases. Define a representative sample and run it on each platform. Make sure the software works correctly in each one, without increasing the time to market.

Implemented Solutions

During 2009 we implemented many solutions that helped us solve the three mentioned challenges at our company:

Google Analytics

Google Analytics allows us to know clients' platforms.

We can obtain details of all visits to our sites: information by country, operating system, browser version, screen colors, screen resolution and Flash, which was used to navigate to our site.

This information is provided in visits and as percentage of the total.

For example, we know what percentage of total users navigate to the site using Internet Explorer 8.0, 7.0, 6.0, etc

Platforms diversity recorded in these measurements is very high, and it is logically impossible to test all. On the other hand, many of those platforms do not have a representative percentage of use. We use the highest percentages to decide the sample to be used for testing the software. For example, if a platform represents 7% or more of the total, the behavior of the site will be considered for testing.

Internal Test Lab Cloud

We implemented the Oracle VM product to generate an internal test lab Cloud.

Through the Oracle VM console, an image of the virtual machine is generated. This image can be cloned as many times as needed according to the hardware resources.

We used each virtual machine to generate the necessary test environments and configured it according to information obtained from Google Analytics.

With Oracle VM we quickly virtualized many testing environments into fewer servers.

The advantage of using this solution is to give independence to the quality control team to manage hardware resources and environments. We avoid dependencies on the availability of new hardware, and can quickly generate test environments.

External Test Lab Cloud

Knowing that the hardware itself has limited resources and that we probably need to get involved with many more environments, we implemented the Amazon Cloud solution. With this solution we can continue configuring more virtual environments into external servers, easily, quickly and without having to invest in hardware. With the same ease of use given by virtual environments to generate in our internal Cloud, we create environments in the Amazon Cloud with the possibility to turn them on and off at any time, paying only for their use.

Define the Test Cases

The next challenge comes from taking all testing environments configured with different platforms. We need to generate all possible test cases and run them in each of the environments without affecting time to market.

We identify the different variables that may exist in each test case and the possible values that each of these variables can have. Af-

ter that we generate the possible combinations of these values.

To achieve this, we implemented an application developed in Ruby that allows us to load the different variables and their values, and it automatically generates all possible combinations

In many cases, this practice creates a very large array of test cases, which is impossible to run considering the time and resources availability we have.

Automated Testing

To resolve this, we developed a Framework to automate test cases. The development was done in "Ruby On Rails". This automation framework allows any member of the Quality Assurance team to automate their tests and run them in any environment.

The QC Analyst may record the transaction that he wants to automate, then upload the script to the framework through an interface. Automatically, the framework manually identifies the values entered by the user and defines them as input variables. In that way the analyst can load all the values that he wants for testing automatically. Through a control panel he can select the test environments to be used for the automatic test run and after completion view a full log of results.

With this solution we currently run a large number of test cases in parallel, each in virtualized environments. With this procedure, we ensure that the same case behaves correctly in each platform without increasing the time to market.

Business Information and Risk Analysis

We can`t always automate all test cases. In these situations, we use two practices that help us to select an effective sample of manual cases. We can use business information to know the actual values that variables can assume. Based on this information we can eliminate those combinations of values that are not feasible in the real world and discard large numbers of meaningless test cases. To achieve this, we use the reports of Data Warehouse (DW).

For example, if we have to test the payment process in all countries and define the field $ as one of the variables to consider, possible values for each variable include decimals. From DW report we discover that in some countries the decimals aren`t used in the $ field. This information will allow us to eliminate many combinations of possible cases and reduce testing time.

The other practice is to apply risk analysis to the set of test cases. It means to analyze the risk associated with a test case to decide whether it is important to run or if it can be eliminated.

The risk is defined based on business impact: economic loss, image and user experience.

It orders the different situations based on risk, so that test cases can be chosen that cover such situations.

Using these practices, we can prioritize test cases based on risk and eliminate the cases with little or zero risk.

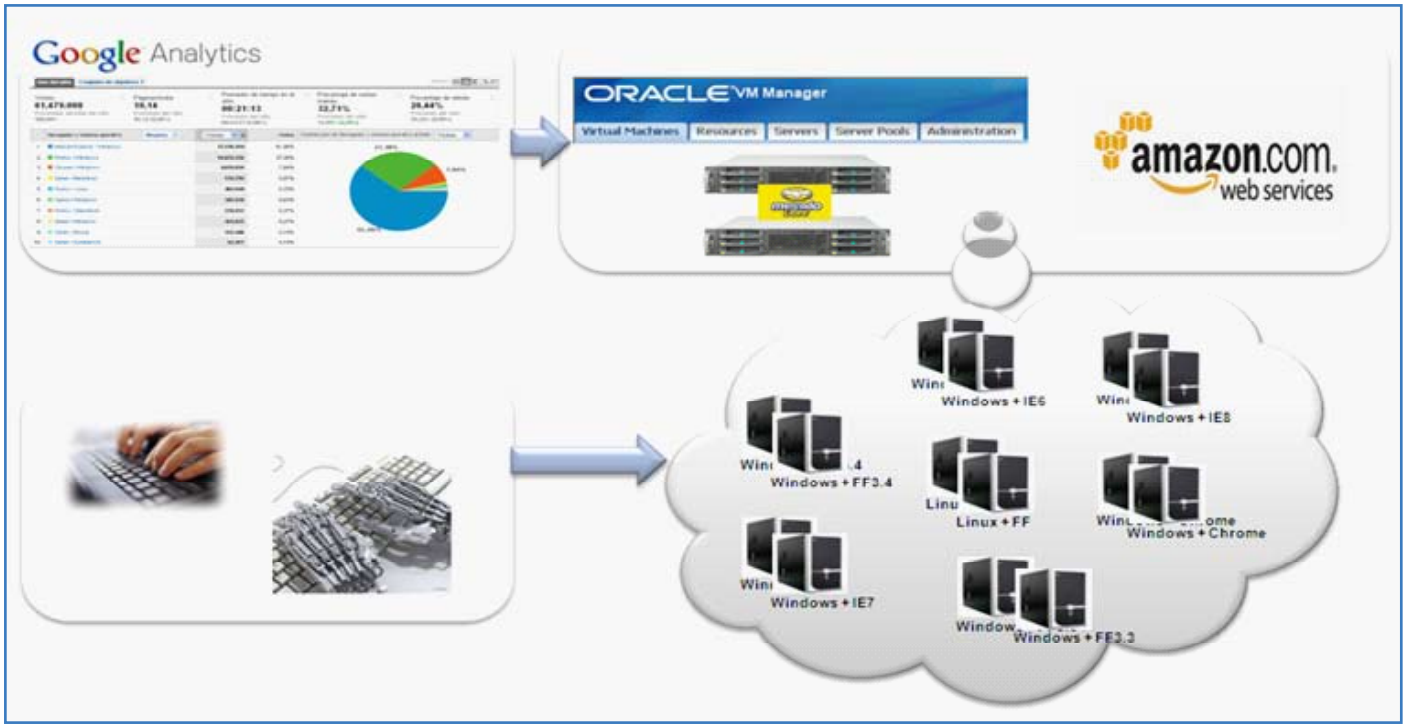
For example, if we have to test a search process, we need to define all possible negative situations (e.g. slowness in the response time of the page, zero result searches, etc) that would cause greatest business impact if they occur. These situations have a high degree of risk associated, and appropriate test cases should be executed.

Results

Implementing all these solutions has allowed us to achieve the following results:

- Create a cloud services platform that provides testing enviroments and testing automation to the Quality Control Team.
- Reduce the hardware's bottleneck to set up test environments.
- Scale quickly and without spending much money on hardware.
- Autonomy of the quality control team to configure and manage hardware resources and the various test environments.
- Ensure the compatibility of the site and new developments in multiple and most important platforms for the clients.
- Ensure site usability and improved user experience.
- Increase test coverage without impacting time to market.
- Meet the challenge of testing the software exactly as it behaves under the complex world of the Internet.

Summary of the solutions implemented



Biography

Rodrigo Guzman (35 years old) is Quality Assurance Senior Manager at MercadoLibre.com, a Latin American leading e-commerce technology company. He joined the company in 2004 and is responsible for defining, implementing and managing software quality policies that enable IT to ensure and control the operation of the website in the 12 countries of Latin America.
Before joining MercadoLibre.com, he served for 10 years in the IT area at Telecom Argentina, a telecommunications company,.
He holds a degree in Business Administration and post degree in Quality and Management. He has fifteen years experience in systems, mainly in processes, projects and quality.
Among other things, he implemented in 2007 the quality management system for the QA department and is certified under ISO9001-2000 standards currently in force. He has also implemented test automation, project management process improvements and agile practices in IT projects.

Guest Column

Challenges in Testing Cloud and Virtual Systems

by Mithun Kumar S R

Cloud computing has already started to prove that it wasn't just a hype in offering tantalizing promises: cost savings, scalability, unlimited storage and being environmentally friendly with the reduced resources.

Spending on IT cloud services is expected to grow nearly threefold, to $42 billion by 2012. Cloud computing will also account for 25% of IT spending growth in 2012, and nearly a third of the IT spending growth in 2013, according to market research firm IDC.

The increased usage comes with an increased risk in various aspectslike functionality, security, operability, reliability and so on. Testing for Risk Mitigation is getting more complex each day, as the dependency on the cloud for business needs soars. Here are a few challenges which are currently being faced when testing such applications.

Test Strategy

Levels and types of testing, which are strategically based on the conventional knowledge of the different software models, do need in addition a multi-level strategy, which includes continuous integration and extensive regression testing. A reactive approach should also be given considerable space, both in the test schedule and plan, along with the proactive approach.

Dynamic Nature of Application Environment

Infrastructure used in cloud changes dynamically, since the resources are shared. This increases the problem of simulation of the environment and calls for extensive automation to improvize on the coverage of all the possible scenarios. Infrastructures failing at run time on multiple accesses are one of the major problems which need to be tackled.

Test Design Techniques

The regular techniques, like Boundary Value Analysis, Equivalence Partitioning, State Transition, or for that matter any Black-Box or White-Box technique, are, however, inevitable for most of the scenarios, but they still leave room for uncovered areas. This would mean that newer techniques need to be applied like that of Musa's Operational profiles.

Continuous Integration Testing

Continuous integration testing should be emphasized more than the traditional integration testing at component level and then integration level. End-to-end interaction becomes critical and day-to-day integration testing should be carried out.

Need for better Emulators

Current testing tools are still not capable of delivering the patterns in traffic, for those accessing the cloud from different geographical locations and over different networks, etc. This poses problems when handling such real-time scenarios used in reality. Major work needs to be achieved in this respect.

Security Testing

Geographical disperse of the resources poses the risk of dreaded attacks on the infrastructure. Exhaustive testing in terms of all the real-world attacks becomes the mantra. Also, testing the restrictions in a multi-tenant application to ensure that users access only their data is critical. The compliance factor also plays an important role.

Skillset of the Test Engineer

Many projects, even those with conventional applications, overrun their budget and time, because of the mismatch in the skillset of the test engineers. The problem becomes worse in the case of Cloud, since a majority of the requirements are not properly documented or are overlooked by the testers. Sound technical know-how of the product is an absolute must for testers in such applications.

The testing challenges seem complex, but would be interesting and rewarding for all those who are willing to take required steps in the revolution.

Mithun Kumar S R works with Tech Mahindra Limited for one of the major global Telecom giants. He holds a Mechanical Engineering degree and is currently pursuing his Masters in Business Laws from NLSIU. As ISTQB Certified Advanced Level Test Manager he regularly conducts training sessions for ISTQB certification aspirants.

Mobile Website Optimization using Multivariate Testing

by Yeshwanta Hebalkar

It was very frustrating just a decade and a half ago to establish net connectivity for offices in a small town in India away from the hustle and bustle of the typical busy metro. The objective was to establish dial-up connectivity to the **National Informatics Center's** district office in the town. It was a lot of blood & sweat with typical text command prompts using UUCP connection and modems buzzing in in the background. Incidentally, at that time the telecom boom was about sending SMSs using pagers. Today, with the exponential growth of net and associated digital technologies, its penetration into our lives is mind-boggling. Proliferation of mobile devices has created a phenomenal impact on our lives due to their multi-functional potential and reach.

The convergence of computing, net with mobile telephony services and its evolution is amazing, leading to the smartphone generation. Typically, smartphone features are touch screen, GPS, accelerometer, camera, net, and keypad. Refer to the recent launch of Google's Nexus One. It has *voice support for all text fields*. You can now dictate your Tweets, Facebook Updates, and emails.

Among the smartphones, typically Apple's iPhone is considered as the game changer. This is because it has addressed many of the problems of the mobile ecosystem from how people interact with the phones, to where they buy phones, to what type of apps users will pay money for, to level of tech standards we can support on constrained devices. Apple has done it and others such as Nokia, Blackberry, Microsoft and Google etc are following the race.

1. Why mobile applications?

Have we ever thought that we would be connecting palm-sized projectors to our smartphones? First came the smartphones, then came apps for smartphones. Now, the next wave of innovation has begun, with the introduction of smartphone accessories that are controlled by apps. This opens up a whole new accessories market and new opportunities to enterprising engineers.

Refer to the December 2009 report from Gartner. Mobile phone users will spend $6.2 billion downloading applications in the year 2010, up from $4.2 billion last year. And the trend appears to only be starting with Gartner forecasting sales in mobile application stores of about $29.5 billion by the end of 2013. These are just the paid applications. The vast majority of downloads, an estimated 82 percent, are free to consumers.

Mobile Application Stores' Number and Revenue, Worldwide

	2009	2010	2013
Downloads (in M)	2,516	4,507	21,646
Total revenue (in $M)	4,237.80	6,770.40	29,479.30

Table 1 (Source: Gartner, December 2009)

2. Context for Mobile applications

For mobile applications, the context is very important. Supposing, a sentence is typed in the text editor, then the editor will, after noticing a full stop, make the next character in the upper case. Similarly, if a driver is driving in a car on the highway, then the mobile device should, without much disruption, be able to provide the meaningful content precisely and quickly. This is where the mobile devices are different to their desktop counterparts. Merely copying or porting the application from the desktop into the mobile domain will not serve the purpose of the user. The context for each mobile user can be different. The user pays for the content while being on the move, for each GPRS packet that gets transacted. For the business user, time is critical, and he cannot afford to spend time to retrieve desired details by browsing pages after pages like with a desktop. Similarly, if the movie option or pizza outlet is to be explored in your local city, then there is no point in having details of movies in some other city. With GPS, the mobile device can understand the present location, which can help in providing movie details within the local city.

3. Mobile apps classification

The mobile applications can be classified broadly as:

- Platform applications
- Web applications
- Mobile websites
- SMS

3.1. Platform applications

The platform or native applications are developed specifically for the target CPU that the mobile device contains. The common platforms are Brew, Palm, Blackberry, .NET, Java ME, Linux, Android,

Mac OS, and Symbian etc. The development of platform applications is relatively simple. However, platform application development, testing & supporting of multiple-device platforms is extremely costly. This limits the product scalability that can impact negatively on return on investment.

3.2. Web applications

The mobile web application is the most acceptable solution to the above problem of development & deployment for providing multi-device support. Mobile web browser is an application that renders content on the mobile device. It is independent of device, platform and OS.

The web browser knows its limitations which enables content to adapt across multiple screen sizes, including orientation either portrait or landscape that is detected by the accelerometer.

3.3. Mobile websites

Due to the mobility context, it is critical to have simple and accurate information rather than an overload of tons of info, which has a bearing on time & cost. The acceptable mobile site segments are email, news, weather, sports, city guides, and social networks. The user seeks the information relevant to meet the individual needs depending on the task and location. In this regard, the personal context is different from the desktop web.

Mobile web access today suffers from interoperability and usability problems. Interoperability issues stem from the device fragmentation of mobile devices, mobile operating systems, and browsers. Usability problems are centered around the small physical size of the mobile form factor, limited resolution screens and user input operating limitations.

Mobile websites are required to be optimized for the content based on the constraints on mobile devices which are listed below.

- **Lack of Javascript and cookies** – Excluding smartphones, most devices do not support client-side scripting and storage of cookies, which are now widely used in most websites for enhancing user experience, facilitating the validation of data entered by the page visitor, etc.
- **Small screen size** – This makes it difficult or impossible to see text and graphics dependent on the standard size of a desktop computer screen.
- **Lack of windows** – On a desktop computer, the ability to open more than one window at a time allows for multi-tasking and for easy reversion to a previous page. On the mobile web, only one page can be displayed at a time, and pages can only be viewed in the sequence they were originally accessed.
- **Navigation** – Most mobile devices do not use a mouse-like pointer, but rather simply an up & down function for scrolling, thereby limiting the flexibility in navigation.
- **Types of pages accessible** – Many sites that can be accessed on a desktop cannot be accessed on a mobile device. Many devices cannot access pages with a secured connection, Flash or other similar software, PDFs, or video sites, although this has recently been changing.
- **Speed** – On most mobile devices, the speed of service is very slow, often slower than dial-up Internet access.
- **Broken pages** – On many devices, a single page as viewed on a desktop is broken into segments, which are each treated as a separate page. Paired with the slow speed, navigation between these pages is slow.
- **Compressed pages** – Many pages, in their conversion to mobile format, are squeezed into an order different from how they would customarily be viewed on a desktop computer.
- **Cost** – The access and bandwidth charges levied by cellphone networks are much, much higher than those for fixed-line internet access.

3.4. SMS

The SMS is the most primitive service that is available to the users of the mobile device. In its simple form, the user sends a query in the form of short code, in order to receive instantaneously the desired information, such as train or flight times, stock prices, savings account balance, etc. The only limitation is that it is text-based, and there is a 160 characters limit on the message content. It can also be used to trigger some tasks or activities, such as dispensing tea or coffee from the vending machine.

4. Challenges

Some of the challenges have been discussed in the above paragraphs. The consolidated list is as follows:

- Service provider or operator control on cost
- Device fragmentation
- Compatibility issues to support different Operating Systems (OS) & development platforms
- - Inconsistencies in user experience due to incompatibility XHTML, CSS, JavaScript to view the same code across browsers
- Browsers are still evolving: Not every browser is able to support mobile web standards and industry best practices

Service Provider or operator control on cost – Boon or bane?

Service providers' or operators' specific role is to provide the wireless service over the cellular network. They need to establish, operate, provide 24/7 support, maintain the network in order to provide reliable service to the end users. They need to interact & maintain their relationship with subscribers. They have to do provisioning, handling, billing of the services. Additionally, they are forced to provide packaged devices – meaning they need to provide support for

mobile device + content + service.

For this service providers go for a tie-up with mobile device manufacturers. This creates competition among the service providers with many plans and campaign offers. Ultimately, this results in putting pressure on service providers to meet unrealistic requirements, such as having to support too many devices in the device family and compliance to relevant certification processes. This eventually has an undesirable effect on the pricing structure. In packaging the contents of mobile applications, they are expected to be provided almost free of charge. This leads to undesirable effects on development resulting into fear of being driven out of business. Typically, the devices are sold at much lower prices with up to 60% discount.

5. Best practices, industry norms such as W3C, .mobi

The World Wide Web Consortium (W3C) mobile web initiative was set up by the W3C to develop best practices and technologies relevant to the mobile web. The goal of the initiative is to make browsing the web from mobile devices more reliable and accessible. The main aim is to evolve standards of data formats for Internet providers that are tailored to the specifications of particular mobile devices. The W3C has published guidelines (Best Practices, Best Practices Checker Software Tool) for mobile content and is actively addressing the problem of device diversity by establishing a technology to support a repository of "Device Descriptions".

5.1. mobileOK

W3C is also developing a validation scheme to assess the readiness of content for the mobile web, through its mobileOK scheme, which will help content developers to quickly determine if their content is web-ready. The W3C guidelines and mobileOK ap-

Es gibt nur einen Weg zum Glück.

gebrauchtwagen.de
Autos zum Verlieben.

proach puts the emphasis on adaptation, which is now seen as the key process in achieving the ubiquitous web, when combined with a Device Description Repository.

5.2. MobiWeb2.0

MobiWeb2.0 addresses the usability and interoperability issues that are holding back mobile web access today. It will focus on mobile Web 2.0 applications based on technologies such as Ajax that can significantly increase the usability of mobile Web applications.

5.3. MobiReady

mTLD Mobile Top Level Domain, Ltd., usually known as dotMobi, is the Internet Corporation for Assigned Names and Numbers (ICANN), the appointed global registry for the .mobi domain. Backed by 13 leading mobile and Internet organizations, .mobi addresses the need for seamless access to the Internet from mobile phones.

DotMobi is the first and only top-level domain dedicated to users who access the Internet through their mobile phones. With four mobile phones purchased for every one personal computer, there's a world of people for whom the mobile phone is the main access point to the Internet. And those users can trust that a website is compatible with their mobile phone if the site's address ends in .mobi.

mTLD, the registry for .mobi, has released a free testing tool called the MobiReady Report to analyze the mobile readiness of websites. It does a free page analysis and gives a Mobi Ready score. This report tests the mobile-readiness of the site using industry best practices & standards. The single page test provides:

- dotMobi compliance
- W3C mobileOK tests
- Device emulators
- Detailed error reports

6. Test strategy – Multivariate Testing (Multivariate Testing - Why, what and how?)

Why - Multivariate Testing?

It has been described that developing and deploying native platform applications for each device is an extremely costly affair. The simplest solution to the problem are web applications. Web applications can be independent of platform, OS and browser. It is relatively easier to support the device families. However, testing has to be conducted for every individual device in order to check the rendering of the application on the device, which varies in screen size. In view of this, the content that is being provided by the mobile website is the key. The context to the mobile user is extremely important and has a personal relevance, which has unique significance in providing the mobile application. The rendering and the amount of data and the associated time are very critical. This is going to be taxing to the end user if it has not been designed correctly. From this perspective, it is important that the website shall provide the information to the user based on his needs and expectations. You cannot afford to assume anything about the user. It is expected that the high-end device experience must be provided in the same manner to the lowest featured device in the family. The judgement of the developer and or the service provider is less important. The mobile is the most personal mass media, unlike any other media such as radio, TV, movie, print media etc. It is to be noted that the mobile device contains almost all the previously established mass media incorporated into one single media. In view of this, it is proposed to optimize the mobile website using Multivariate Testing.

The only way to figure out what content will work best on the mobile website is to test different content. Multivariate test strategy will help you study the effects of different content on your users. Testing will enable you to identify what users respond to best, so you can create a website that will be more effective in getting the business results you want.

What - Multivariate Testing?

A/B Testing Vs. Multivariate Testing

A/B testing is used to test a few different versions of one single element on a page, e.g. title, font, background color, image, size of image, border thickness, text content etc. by creating variations in different page version. It will contain one reference page and one or multiple content variation pages. It is possible to conduct A/B tests with just two pages as a minimum requirement. A/B tests require a few hundred conversions taking place over a maximum of several weeks for conclusive results in order to determine the winning combination.

It is possible that at the end of the A/B test there may not be a conclusive result. If the results are inconclusive, this may mean that we weren't able to sample a significant number of users, or there wasn't a significant difference between the combinations tested. If there wasn't a significant difference between combinations, but several combinations performed better than your original content, you may wish to replace the original content on your live site with the HTML content for one of the better-performing combinations. Alternatively, you may prefer to edit and run your experiment again. Thus it will be an iterative process to conclude based on the goals and objectives of the test. In this manner it will be an adaptive process that fits into **Agile methodology**. It means do not create big changes, even with the small changes in the iterative manner the site optimization experiment can be conducted for achieving the success of the website.

Multivariate testing is a process by which more than one component of a website may be tested in a live environment. It can be thought of in simple terms as multiple A/B tests performed on one page at the same time. A/B tests are usually performed to determine the better of two content variations, multivariate testing can theoretically test the effectiveness of limitless combinations. The only limits on the number of combinations and the number of variables in a multivariate test are the amount of time it will take to get a statistically valid sample of visitors and computational power.

Multivariate testing is usually employed in order to ascertain which content or creative variation produces the best improvement in the defined goals of a website, whether that be user registrations or successful completion of a checkout process.

How - Multivariate Testing?

There are two principal approaches used to achieve multivariate testing on websites.

- Page Tagging
- By establishing a DNS-proxy or hosting within a website's own data center

Page Tagging is a process where the website creator inserts JavaScript into the site to inject content variants and monitor visitor response. Page tagging usually needs to be done by a technical team and typically cannot be accomplished by a web marketer or a business analyst.

The second approach used does not require page tagging. By establishing a DNS-proxy or hosting within a website's own data center, it is possible to intercept and process all web traffic to and from the site undergoing testing, insert variants and monitor visitor response. In this case, all logic resides in the server rather than browser-side, and after initial DNS changes are made, no further technical involvement is required from the website point of view.

7. Test setup description

In this article for Multivariate Testing sample example is demon-

strated using Page Tagging. As already pointed out, Multivariate Testing is multiple A/B tests performed on one page at the same time.

For example in this case the A/B Test is under consideration. There are many commercial outsourced test providers available. There are proprietary tools available or service is available from service providers. In the present case, an open- source tool from Google is used as an illustration for A/B Testing. The tool used is Google Web Site Optimizer (GWO). In our case the objective of the test was to check the tool usefulness and to demonstrate the mobile website optimization test result using GWO.

Testing is carried out on a hosted mobile website by setting up the server to display the different variations of content, such as heading, different images, image sizes, background color, etc. to incoming visitors.

Mobile website hosting

There are in total three web pages for the .mobi website. The home page is the reference page. The alternate page is designed to create the variation in the reference page. The third page is the conversion page, which is just the reference conversion displaying a thank you message.

The reference to the hosted mobile website is provided as below.

http://hebsyesh.mobi/hebs%20home.htm

The tagged scripts are added to the web pages as per the GWO tool-generated JavaScript. The uploaded pages contain these tagged scripts. The web page testing is conducted using the mobiReady online tool. http://ready.mobi/launch.jsp?locale=en_EN

The test report summary is as below.

It will probably display well on a mobile phone.

Your mobi.readiness score is calculated from the results displayed below. Failing tests and large page sizes both lower the score. Read through the report to understand how to improve your score - and your site.

Test summary

20 Passes	2 Warnings	4 Fails	2 Comments

GWO Tool Report

Statistics about total number of visitors to each page is measured by the GWO. It also indicates the total number of conversions.

The report page is shown as an illustration.

The report table indicates the original reference page visitors as 23 and conversions as 19. For Variation1, the visitors are 21 and conversions are 16. It still continues to collect the data in order to get a conclusive result.

8. Testing challenges

Some challenges were encountered:

- Hosting the .mobi web site
- Testing the website with adequate user load

The GWO recognizes visitors with user visit if they are separated by distinct IP addresses. There are commercial and open-source performance testing tools available in order to create virtual user loads. The commercial tool access and availability is the challenge. Skill is required to do the needed code configuration. However, to create the user load in the order of 500 user,s the support for IP spoofing is required, which needs specific skills.

If you visit once the web page for the configured website under GWO, it takes sometimes a few hours to count and reflect it in the GWO report as a visitor and converted page if applicable.

9. Test results & analysis

Refer to the mobiReady mobile website test tool test report. It has conducted testing

- dotMobi compliance
- W3C mobileOK tests
- Device emulators
- Detailed error reports

There are 4 failures listed below. The detailed failure description is available in the online report.

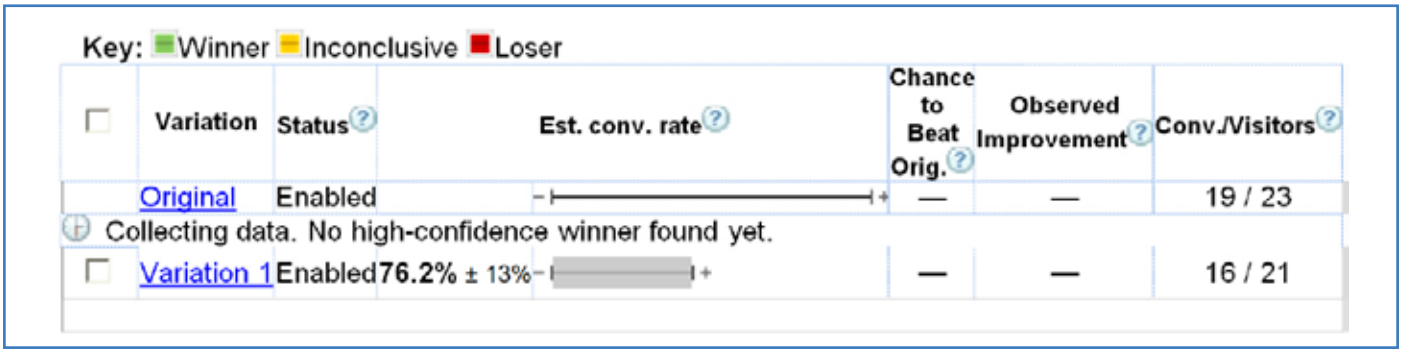
- Your page does not use XHTML Mobile Profile
- Your page markup does not validate
- Incorrect or missing character encoding
- No cache control headers were sent with this page

These can all be fixed based on the detailed description analysis provided in the online report.

The GWO tool report is inconclusive at this stage, in order to decide on winner or loser. However, it has provided the conversion rate. The main reason is that no adequate number of visitors is available at this stage. It should be noted that this is due to resource constraints. The testing continues till the sizeable number of visitors in the order of 500 is counted.

10. Conclusion & recommendations

Multivariate testing is currently an area of high growth in Internet marketing, as it helps website owners to ensure that they are getting the most from the visitors arriving at their site. Areas such



as search engine optimization and pay-per-click advertising bring visitors to a site and have been extensively used by many organizations, but multivariate testing allows Internet marketers to ensure that visitors are shown the right offers, content and layout to convert them to sale, registration or the desired action, once they have arrived at the website.

Multivariate testing can also be very useful in other scenarios, for example to conduct opinion polls, free online services & applications survey, online print media, non-governmental organizations for understanding needs of targeted audiences within the society.

Multivariate testing for mobile websites

It is concluded that mobile and computing aspects can be effectively addressed with optimized mobile website design as a part of user acceptance tests. In this context, the innovation is user-driven rather than by trying to sell the next big thing. Business analysts and marketing professionals have to play an important role when designing the test, understand the user context covering different geographic locations and optimize the tests. This is an iterative exercise. This really can be decisive for the value added to users. It is also important to find out which portion of the web content is not accepted by the user, so that it can be avoided and lessen the time & cost burden. Ultimately, it is realized that multivariate testing is the most effective scientific technique to design websites based on the user expectations. Mobile web analytics is an emerging area in the mobile net domain.

It is strongly recommended to adopt multivariate testing at the user acceptance level in the lifecycle model for mobile website optimization. This can be an ongoing exercise if true potential is to be achieved, and in order to stay competitive. It is better to know which mobile content offerings will convince and engage consumers most effectively, and then act upon the results. Iterative incremental changes can produce the best results, which is the best fit under Agile methodology.

References:

1. Mobile Design and Development by Brian Fling; Publisher: O'Reilly
2. Gartner press release Dec 2009 http://www.gartner.com/it/page.jsp?id=1282413

   "Gartner Says Consumers Will Spend $6.2 Billion in Mobile Application Stores in 2010"
3. Mobile Web http://en.wikipedia.org/wiki/Mobile_Web
4. Effective smartphone accessory design http://www.embedded.com/222400483?cid=NL_embedded
5. Mobile Web Optimization Via Multivariate Testing http://www.mediapost.com/publications/?fa=Articles.showArticle&art_aid=83319
6. mobiReady Online Tool http://ready.mobi/launch.jsp?locale=en_EN
7. Google Website Optimizer Online Tool https://www.google.com/accounts/ServiceLogin?service=websiteoptimizer&continue=http://www.google.com/analytics/siteopt/%3Fhl%3Den&hl=en

Biography

Yeshwanta Hebalkar works at Mahindra Satyam's Group / Satyam Computer Services Ltd., Hyderabad, India. Currently, he is working as Consultant in various IT projects and programs. In the past, he worked with Invensys India Pvt Ltd., Danlaw Technologies India Ltd. His interests are to provide vertical-based solutions using automated combinatorial, multivariate testing and web analytics. He holds a Masters degree in Electronics as major from Fergusson College, affiliated to Pune University India. He started his career as senior research assistant in IIT Mumbai. He has experience of two decades in Telecom & Radar, Defense, Automotive and Consumer Electronics industries. He was awarded fellowship in Electronics and subsequently worked as Scientist in Defense R & D Organization from 1988 to 1999. He is the recipient of an award from Scientific Advisor in recognition to the significant contributions to phased array radar.

I'm your New Project Manager!!!

by Allmas Mullah & Ajay Balamurugadas

Synopsis

This article describes how the authors have applied the learnings from different schools of testing to a testing problem in their day-to-day testing activities. They emphasize the fact that there is no ONE school of testing which suits all the contexts. The authors take you through a journey of a manager helping the team solve their testing issues.

Welcome Aboard

"We expect a lot from you!"... "Sure! Everyone does!" Carla thought smiling to herself, when Ted, her new delivery manager had set his expectations upfront on her first day as the test manager.

He told her "We've had issues with the quality of the software delivered to customers and complaints from them resulting in some serious money being used to clean up the act. The management is not happy with our efforts..." and he added, "Remember, our jobs depend on the forthcoming results of the customer satisfaction survey.....which is in three months!"

Understanding the context

After many short meetings, informal chats over coffee with teammates, Carla realized that the actual problem was hidden below the surface, just like an iceberg. On analyzing the different documents presented to her – bug reports, test suites, weekly reports, customer feedback, appraisal forms, one thing was crystal clear: "The management was playing the numbers game, **and the test team fell into their numbers trap**"

Knowing the rules of the game

After she gathered enough data to identify the issues, Carla designed a mind map and arranged for a meeting with the team to discuss the issues.

She identified some of the key points as follows:

➤ **Relationship with programming team**

According to the programming team, this testing team was inefficient. The facts and figures favored the programming team.

There was a higher number of invalid bugs, of non-reproducible bugs, and only relatively few valid bugs being reported. One major complaint against the testing team was "Discovery of bugs just before release".

➤ **Testers' understanding about testing**

The current testing team had a narrow view of what "testing" meant. They were happy with just reporting as many bugs as possible. They failed to identify the traps in software testing [1] and the importance of educating the internal and external customers. Bug Advocacy was a skill unknown to the test team.

➤ **Management's expectation from testers**

It was no surprise that management found it easy to impose their numbers game on this team.

The management theory was simple: Testers should complete the test suite, file bugs, and submit status reports with numbers.

Testing was treated as a mechanical process. Neither the management nor the testers themselves thought about testing as an intellectual art.

Management regarded the colorful charts and graphs as an important deliverable.

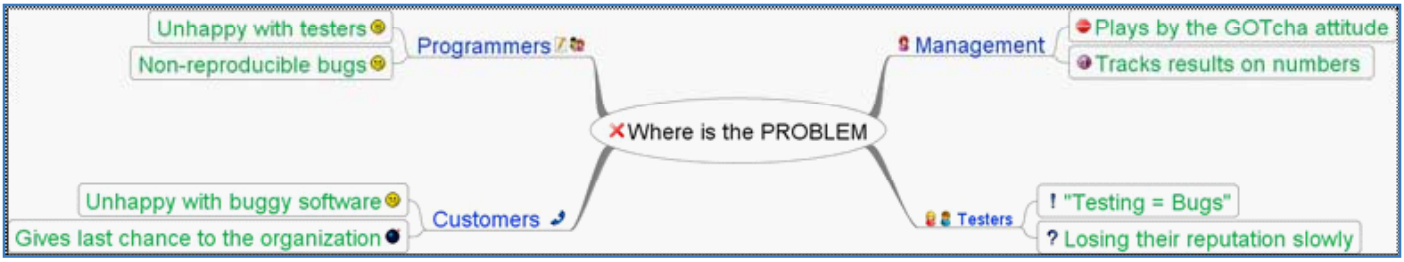


Figure 1. Where is the Problem

➤ **Fulfilling customer requirements**

Customers who had no clue of the inside story were the victims of the buggy software. To rub salt on the customers' wounds, they had to deal with delayed releases. The customers were furious and their anger was justified. It was clearly reflected in the poor feedback survey results.

Carla decides to be hands-on

Carla chalked out a plan that would enlighten and empower the testers.

Some of the testers including Raj, the test lead, had some concerns about the new changes. But then, they had no other option, as they were staring at a dead end. This was one of the escape routes.

Her plan of action was simple; utilize the strengths of EVERY individual to accomplish the task.

Any team is a mixed bag of talent. Carla's team was no different. Some were good at analysis, some at writing tests, and some at using programming languages, while some were smart bug hunters. Forcing every individual to do the same task ignoring their skillset resulted in the current state of the team. Every tester had to go through the routine of opening a test suite, running the same tests build after build, and updating the test results. There was neither any new learning nor excitement in their testing activities.

Keeping the three-month deadline in her mind, she along with Raj took every tester in confidence.

Carla requested Ted to stop monitoring this team for two weeks and Ted reluctantly agreed.

Her approach focused on three main dimensions:-

Coverage

Requirements were cross-checked with existing test cases. The gaps were identified, and clarifications were obtained from the business analyst and the programming team.

Some of the existing test cases were updated. Some new cases were also written to achieve requirements coverage. The programming manager was surprised at this new approach in place.

Carla was a follower of the experts of the testing world like Michael Bolton, who has explained the use of tests and checks [3] for effective testing. Carla encouraged her team to use this approach.

Two testers from the team volunteered to automate the smoke test suite. This saved a lot of time, and testers could now focus on critical tests.

Potential Problems

With the confidence of the testers by her side, Carla demonstrated the power of discovering "Potential problems". This had never been done before, as testers were mechanically executing the test cases. As the Potential Problem dimension mentions: "What risk you're testing for?" [2] The enthusiastic team set out hunting for potential problems.

Carla could sense the new energy around her team.

Activities-based techniques

With the "What" and "Why" covered, she set her focus to "How do you test?"

➤ **Scripted Testing**

This proved helpful to test the requirements stated in the specifications. The test suite covered the requirements.

➤ **Exploratory Testing**

Carla's experience in using the exploratory testing approach was of immense benefit to the team. She completed a session of PAIRED testing with each tester and de-briefed about the learning, mistakes and tasks after the session.

Some of the skills which were given importance were:

- Questioning
- Observation
- Taking notes
- Bug-investigation skills
- Critical thinking
- Use of tools and resources

➤ **Long-Sequence testing**

As per Carla's analysis, one of the scenarios which resulted in a number of leaked bugs to the customer was the difference in the test environment. While the test team received weekly builds, the customer received a build once every three weeks. Prolonged usage of the software for three weeks seemed to have a definite impact on the behavior of the software.

Raj assigned one resource to monitor issues arising out of 'Long Sequence Testing'.

These test techniques were employed for two consecutive builds.

The Change in Mindset

More questions, more interactions, increased number of important problems found, more learning and more fun were some of the fruits of Carla's management.

Some of the initial hiccups like testers' ability to tell a convincing testing story to sell the bugs, manage testing activities within the time-boxed session, usage of new tools to aid testing, was slowly offset by practice and observing more experienced testers do it with ease.

To gain some more time for testing, Carla requested Raj to ask testers to stop creating status reports on an hourly basis. She was happy with a weekly report and the session notes by each tester. The testing team was working as a TEAM.

It was the third week and Ted was ready to fire his questions to Carla.

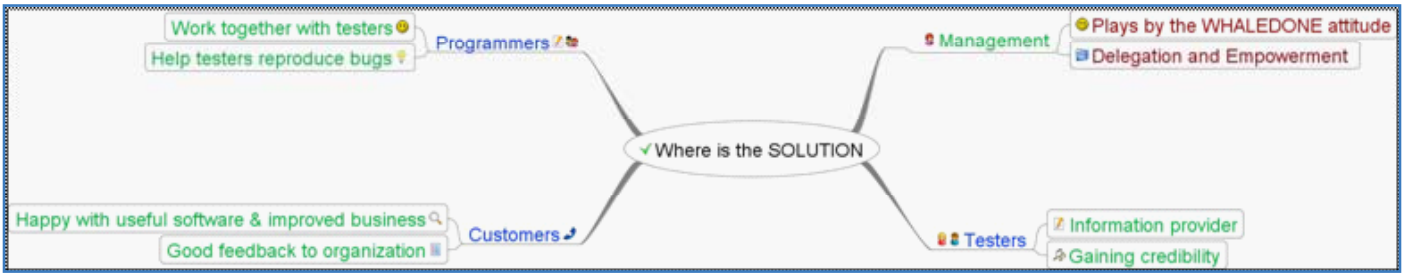


Figure 2. Where is the Solution

Carla gave her usual broad smile and was already prepared with the reports on how the team had effectively spent the two weeks.

Carla had a new mind map to share with the team now.

Conclusion

The smile on the testers' faces was as bright as the summer sun. Testers were actually TESTING and enjoying it. The testers shared their thoughts and experiences with every other tester. Now, they had realized the importance of improving their skills to test better, learning and understanding techniques to help meet the mission.

Carla and her testing team are on a two-day outing as the programming team is busy fixing bugs.

Do you wish Carla was your project manager?

References

[1] http://searchsoftwarequality.techtarget.com/news/article/0,289142,sid92_gci1256314,00.html

[2] Testing vs. Checking - http://www.developsense.com/2009/08/testing-vs-checking.html

[3] Kaner, Cem; Bach, James; Pettichord, Bret (2001). Lessons Learned in Software Testing. John Wiley & Sons. ISBN 0471081124.

Biography

Allmas Mullah is a black-box software tester based in India.
Having worked in the software development field for nearly a decade, she has experienced different approaches of software development, from waterfall to the latest Agile-Scrum.
Allmas has attended Michael Bolton's workshop on 'Rapid Software Testing for test managers and leads' and is now implementing the knowledge gained at the workplace.
She blogs at "http://allmasmullah.blogspot.com/" and tweets @Allmas.

Ajay Balamurugadas, student of Pradeep Soundararjan is a software tester, passionate to learn to test any software. He has been awarded scholarship by the Software Testing Club and is a proud student of Miagi-Do School run by Matt Heusser. He is co-founder of "Weekend Testing". Ajay shares his testing activities and learning through his blog "Enjoy Testing" and tweets @ajay184f.

Subscribe for the printed issue!

Please fax this form to +49 (0)30 74 76 28 99, send an e-mail to info@testingexperience.com or subscribe at www.testingexperience-shop.com:

Billing Adress

Company: _____

VAT ID: _____

First Name: _____

Last Name: _____

Street: _____

Post Code: _____

City, State: _____

Country: _____

Phone/Fax: _____

E-mail: _____

Delivery Address (if differs from the one above)

Company: _____

First Name: _____

Last Name: _____

Street: _____

Post Code: _____

City, State: _____

Country: _____

Phone/Fax: _____

E-mail: _____

Remarks: _____

1 year subscription

☐ **32,- €**
(plus VAT & shipping)

2 years subscription

☐ **60,- €**
(plus VAT & shipping)

Date

Signature, Company Stamp

Enhancing SOA Testing With Virtualization

by Gaurish Hattangadi

What is Service Virtualization?

Service Oriented Architecture (SOA) promises a new paradigm for IT – one that aligns IT with business and provides a range of benefits. SOA evangelists tout system reuse, enhanced application lifetimes, improved integration, IT agility and many other advantages to this methodology. Inspired by this, many IT managers have taken the SOA and Business Process Management (BPM) plunge, only to realize that implementation and testing of systems engineered with SOA isn't that straightforward – for example regression testing of SOA applications is more complex and critical due to the integrated nature. To address these SOA testing challenges, a variety of new tools and approaches have evolved, making service virtualization one of the hottest ideas to hit the market.

What questions does SOA pose, that have led to the rise of virtualization? Here are a few that I have heard from customers:

1. SOA encourages system reuse, but how does a manager enable multiple development teams to develop shared systems?

2. Service orientation encourages integration, but how does an architect test and develop when some services are unavailable for testing?

3. SOA encourages integration, but how do you reduce the cost of testing systems that might have a high per-use licensing model?

Service virtualization is the capability to capture the behavior of a service and replicate it in a virtual environment. Service virtualization allows for expensive, inaccessible and scarce resources/assets to be modeled and replicated in a virtual environment, thus effectively solving the problem of resource availability and cost escalation. Architects can create multiple virtual mock-ups of real-world services and allocate each to a different team, allowing for parallel development on shared systems. IT managers can virtualize expensive services thus reducing project costs. Service virtualization is thus an emerging and indispensible tool in the SOA testing landscape today.

The service virtualization process has three steps:

1. Capture the behavior of the service in the real world.
2. Use the captured behavior and other information to create a mock service and deploy it in the virtual world.
3. Modify the orthogonal aspects of the service when needed e.g. modify response times.

Some product vendors extend the definition of virtualization in the SOA world to also encompass other IT components in solution architecture. This can include virtualization of components including mainframe systems and databases. This extended application of virtualization to the SOA world allows for more sophisticated SOA implementation when these dependent systems have high access/usage costs.

The term "Virtualization" is also widely used in other contexts, such as storage virtualization, memory virtualization and platform virtualization (the Java Runtime Environment and the .NET Common Language Runtime). This paper focuses on the use of virtualization as it applies to the problems of testing modern SOA applications.

The Evolution of Service Virtualization

Virtualization has grown out of the need to optimize and drive down Total Cost of Ownership (TCO). My earliest exposure to virtualization was working on VMware systems, an application of hardware virtualization. The premise of hardware virtualization is simple, underutilized hardware litters IT infrastructure in an organization. Is it possible to use this unused capacity to serve applications? IT managers can use hardware virtualization, which by pooling together hardware resources and allocating this pooled infrastructure to applications drives down TCO. This is a great idea that has wide penetration throughout enterprises worldwide. Can this aspect of virtualization be applied to SOA?

We know that the SOA story promises a wide range of benefits that include agility, lowered costs and better integration. Today SOA is utility IT, something that must be done in order to have lean agile baseline IT. The question isn't really who does it anymore, but who does it better. The better SOA equation is very simple – it's about who develops a quality service layer faster, cheaper and quicker. Service-oriented development and testing has evolved a great deal, we now have sophisticated new SOA development methodologies, testing methodologies and testing tools. While SOA speaks of autonomous systems that are integrated yet free to evolve independently, it fails to decouple the development team that must share access to these systems. Large integration implementation projects are riddled with sequencing of tasks that involve shared systems. This is a very expensive problem. Another problem we see is that service orientation promises better integration, enabling business to buy (vs. develop) services that are not central to a business. For example, a manufacturing firm can choose to use a Software as a Service (SaaS) Customer Relationship Management system (CRM) instead of spending money in developing the application in-house. What this methodology fails to address is the additional money expended in testing the CRM integration due to various factors, including complex data setup. Hardware virtualization does not provide a solution to

these problems. Not all problem systems are amenable to hardware virtualization, an example being mainframe or SaaS systems. Some scenarios such as performance testing are not opportune areas for hardware virtualization. Hardware virtualization targets underutilized systems, where resources are inefficiently used and spare capacity can be found and consumed. While hardware virtualization has its place in the SOA world, the problems that we have seen above deal with bottlenecks, these are over-utilized systems and systems that need behavioral replication.

With service virtualization, the ability to emulate the behavior of a service in a virtual environment is gold when the application has a per-use charge or when the data setup and tear-down can be difficult and time-consuming. Service virtualization is the answer to optimizing bottleneck systems and is a key driver in the push for better, cheaper service layers.

Service virtualization has evolved to encompass not just services but also database systems, mainframes and other components in IT. Some products will allow for complex modeling of component responses based on inputs. A few products even allow for virtual databases to be modified with stored procedures and functions. Service virtualization has progressed to a point where it can significantly affect a project's critical chain in both testing as well as development.

Before and After Service Virtualization

In the SOA world, where everything is integrated, project managers often face an arduous problem of shared access to reused systems. If SOA promises leaner development through reuse, what happens when the systems to be developed are not amenable to co-development between teams? Service virtualization can be used to emulate a service that needs shared access. Imagine a situation where a CRM system needs to be integrated by two teams with varied test data setup and other requirements. Classically, you would need to provide sequential access to the system, thus doubling development and testing time. With service virtualization both teams can have their own virtual service and develop, test independently of each other. Service virtualization now allows the project manager to parallelize development and testing.

We have seen SOA development plagued by another problem that is easily solved by virtualization. Let's assume that a project plan calls for a mainframe to be exposed through services – these services are to be then used by a modern J2EE application. Classical project management would have dictated that the project be done in two phases – phase one executing the mainframe services development and phase two would be the J2EE application. With virtualization, an architect can model the services to be developed by the mainframe team and release the virtual model to the testing team. This is our SOA dream of interoperability – not only services being interoperable in production, but allowing development teams to interoperate in implementation as well. Development can now begin for both teams in parallel.

Service virtualization has another very strong use case – one that solves a problem that has never been easily amenable previously. Let's assume that your project mandates the integration of a service that cannot be exposed easily for testing. Perhaps this is a third-party or partner service that is plagued by issues that make test data setup difficult or time-consuming. Without virtualization, development teams would have been forced to create stubs and hope that all would work well in production. This is dangerous in SOA, where integration is key and ripple effects can devastate a complete process. With virtualization you point your mouse at the service, capture production behavior and violà! You have a replica that can be hosted in your virtual environment, and now your developers and testers have a viable development / testing strategy.

These different use cases for virtualization are all born out of one necessity – easing resource constraints. If resource optimization and targeting resource bottlenecks is the key to project management, then service virtualization is the answer. Parallel development and testing can lead to shorter development and testing time. Service virtualization can make your organization the one with a cheaper and better service layer.

Further Steps in Virtualization

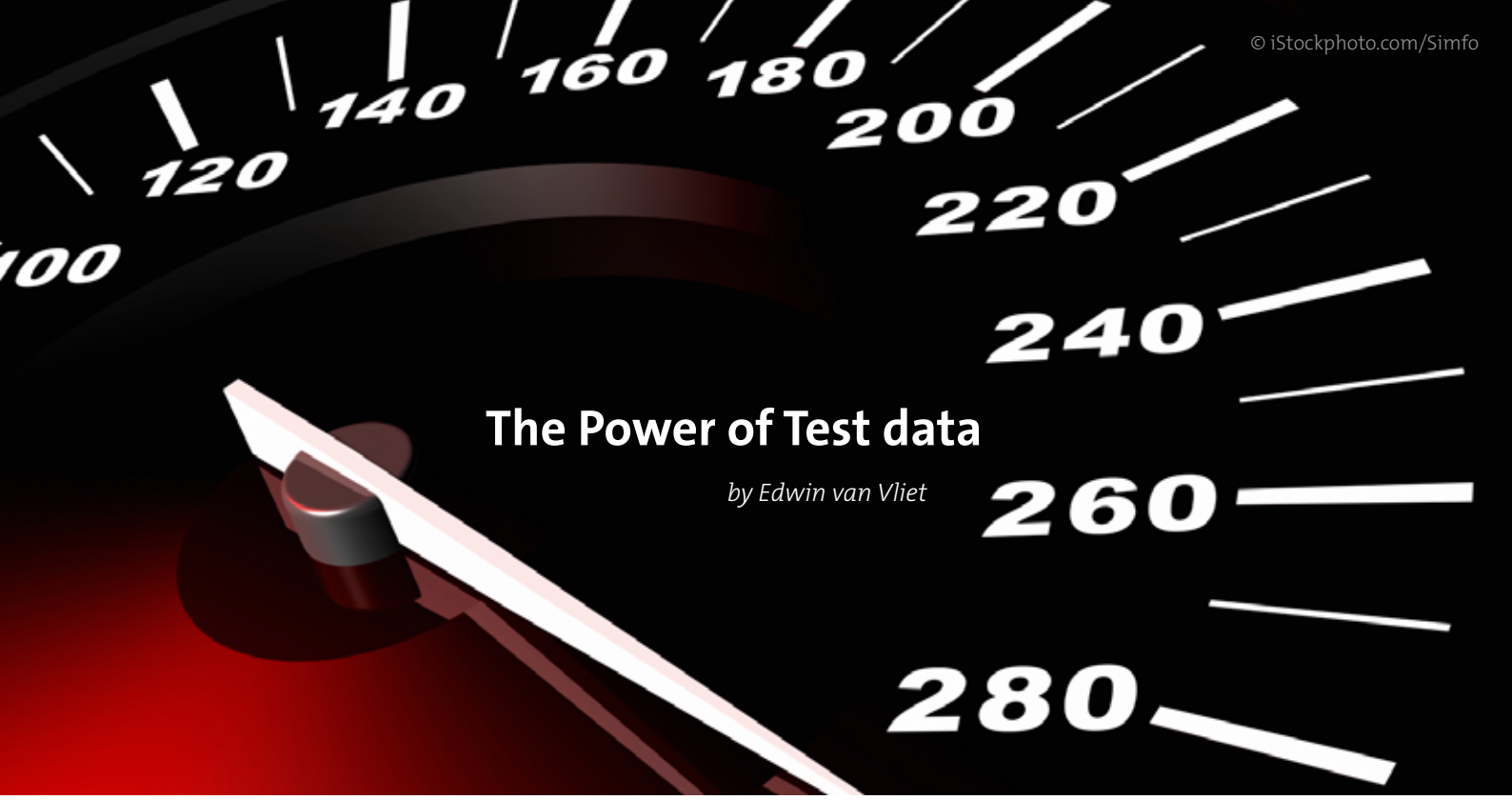
Service virtualization is a promising new development in the SOA testing sphere. Virtualization, when correctly implemented under the guidance of a suitable methodology, provides:

1. A way to optimize usage of scarce resources in development and testing.
2. A way to optimize the project-critical chain, allowing parallel development and testing.
3. A way to emulate a resource in development and testing that is unavailable/ expensive in the real world.

Do remember that different product vendors interpret service virtualization in different ways, ranging from service stubs with fairly primitive behavior to sophisticated tools that can emulate the behavior of major components in the SOA world, including mainframes, services and database systems. Your success in the virtualization story will be driven by your product vendor's capability and implementation methodology.

Biography

Gaurish Hattangadi is a consultant at Infosys Technologies Ltd., where he enjoys promoting and researching SOA testing. He is an ardent student of enterprise architecture and has taught the subject at numerous large software consulting firms.



The Power of Test data

by Edwin van Vliet

In creating or providing a test or development environment, the tester controls most of the activity. In a tester's world, people can be born, get married or die, all their money can be transferred to your account and you can happily play with all aspects of people's information. However, what happens when an accidental error in the infrastructure appears and the test data suddenly reaches the production environment? It has happened before. My point is this: By handling your test data with care in your test environment, you can reduce all damage possible.

According to Gartner, more than 80% of companies are using production or "live" data for non-production activities such as in-house development, outsourced/off-shored development, testing, quality assurance and pilot programs. About 30% of the testers have experienced that test data, by accident, turn up in the production environment.

In the last few years, a trend has emerged where companies are starting to organize their test data management in a more structured manner. The driver for this restructuring is mainly recent privacy regulations, but improving test processes, which are becoming more effective and having a shorter time to market, can be other reasons.

Test data is a fundamental requirement for testing. It is strange that test data management (TDM) is heavily underexposed in test process improvement. This lack of effective TDM is probably due to the complexity and the mixed responsibilities of the organization and results in poorly organized TDM strategies.

Test data is complex

A typical large organization will have at least 10 to 40 back-end systems. Each system will typically have to control its own back-end system as well as cross- connect to other systems. Generic data such as addresses, products and pricing, which can be used by multiple systems, can be set up as separate back-end databases. Synchronizing these large and complex back-end systems is a common nightmare for every tester. The test data need to be present in multiple back-end systems and be cross-domain consistent before a tester can test. All of this data needs to be connected properly before it can be used.

Test data is volatile

When test data is used once, there is a possibility that it can't be re-used. For example, when a tester closes an account in the test data or withdraws a large sum of money from an account, the entry criteria of the test data is not valid anymore for the test execution of a next test run.

Lightning and fire are, like test data, powerful phenomenons.

The power of test data

Test data is like lightning and fire
- If not under control, powerful things can happen.
- If controlled, you are able to do powerful things.

Lightning and fire can burn companies and houses down, incorrect test data can have a similar devastating effect.

Incorrect test data

Test data often contains sensitive and private personal data like addresses, credit card information or health care records. Test data therefore has to comply with privacy regulations. You can take privacy regulations into account by generating fictive or synthetic data or by obfuscating it, like scrambling or mixing the sensitive test data.

Privacy regulations ensure that an individual does not become a victim of misuse of private and sensitive information. For example, when you enter your credit card number for buying a product or service, you should have the certainty that the company you've purchased from will use it with care. Using your personal account details for testing and development is not part of the deal! Therefore, using a direct copy of production data in a test environment is not recommended.

When you use a copy of your production or "live" database to provide test data and this accidentally results in incorrect mutations into production, it can cause loss of customer confidence and personal damage to clients. Loss of customer confidence will affect the company's reputation and can possibly cause bankruptcy. To the clients involved, it can result in long-term problems.

A test result is normally based on testing with test data. When tested with incorrect test data, the test result can give a wrong indication of the software quality. Issues can be incorrectly identified as passed or failed as a result of using wrong test data.

One thing is for sure: testing with incorrect test data will cause unnecessary delays.

Correct test data

If lightning is controlled, we can use its electricity for lighting. When fire is controlled, we are able to use its heat to keep warm and its power to send people into space. The possibilities are endless when lightning and fire are controlled.

These examples can be applied to the use of correct test data. When you use correct test data and start applying test data management strategies, the test data will work for the test team and the possibilities are endless.

Fictive or synthetic test data or depersonalized data is not only used for privacy regulations, but it also becomes a safety net. This safety net prevents damage to the clients and the organization in case test data accidentally reaches production. In case you use your own address information, it will also become a trigger that test data is arriving in production environment unintentionally.

Correct test data will reduce the time to market. The lead time of the test project is reduced by about 40% when the correct test data is present at the moment the test team requires it, and the test data meets their requirements.

Using effective TDM strategies will aid the test team, so they do not have to wait until the data is entered in the test environment. When you organize sufficient test data for all possible test runs, you will not introduce any delays in the next test runs.

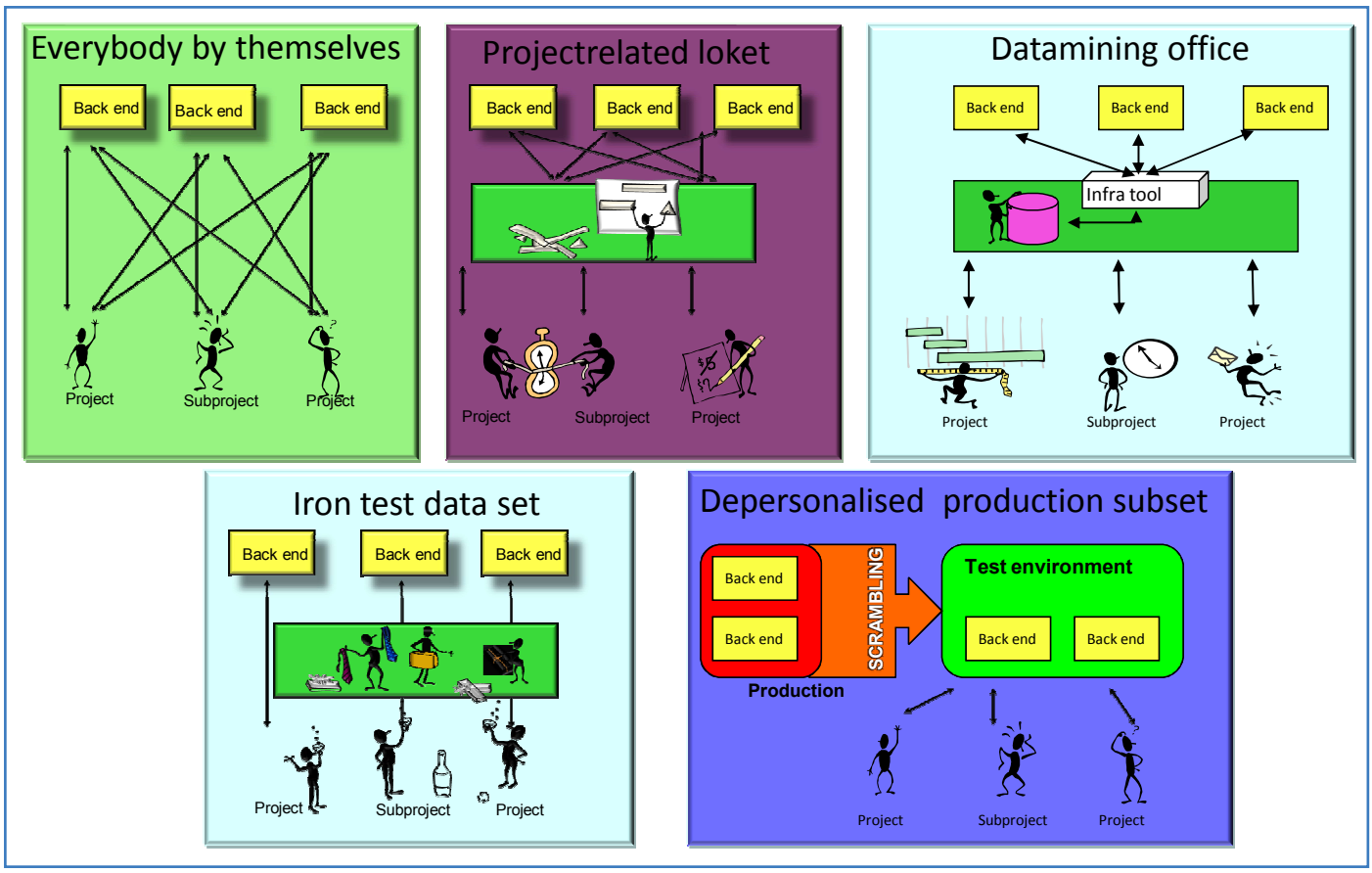
Correct test data enhances the use of test techniques and considers implicit testing. Why should it be necessary to provide minimal data requests and then modify data on the fly during test execution? What can also be done is to create new test data per test case. That way the tester only needs to start-up the test with the correct test data, and the test execution is nearly finished.

A test data request is usually based on some specific fields (like gender, kind of account, type of product). A back-end system requires more fields than the data request. The remaining fields need to be filled with some data, that will not be used for the specific test case.

On the other hand, these remaining fields can be filled with interesting data, like semantics and syntactically correct data.

In this case you are testing the normal test case as required, but you are also testing other fields of syntax-testing and semantic-testing implicitly, without any additional effort.

There is no reason anymore for any organization not to have good test data management practices. In the last 5-6 years, many good tools have appeared in the market like Grid-Tools, Micro focus, IBM, HP, Rational and others. Some of these tools can generate and create correct, fictive and cross-domain consistent test data.



Every organization, without exception, can organize its test data management and make use of the positive power of test data. Just like no individual or organization is the same, there is no standard solution for test data management. For example, scrambling of production data may not be the ultimate solution for many organizations. There are many factors that will influence the decision for the right solution for your organization. Bear in mind the current and future situation of the company organization, test organization, test environments, number of back-end systems, infrastructure and the purpose of test data management.

There are 5 generic test data types to setup test data management within organizations.

Everybody by themselves

Everybody organizes their own test data for their own project. This will result in heroes who have all kinds of interesting lists and test data built up over time.

Project-related office

Test data will be generated by a central team for every project that fulfils only the project-related test data requirements. When the project is finished, this test data can be removed from the back-end systems to keep the test environment manageable.

Data mining office

By making use of (web) services, it is possible to see beforehand which data is available in the back-end systems. If this information is collected in a central repository, the company is able to indicate which test data in the back-end complies to the requested test data profiles. Data requests and results can be compared over time, offering the ability to quickly identify if anything has changed.

Iron test data set

In small test environments it might be interesting to create a test data set which contains all possible combinations of test data. You are able to set this up once and restore it whenever required. At the time when the test team needs fresh data, the iron set can be restored.

Depersonalized production subset

An often used test data type is to retrieve a subset with consistent data out of the production environment and depersonalize this during the extraction. This depersonalized data can be placed in a test environment.

Also varieties and combinations of these 5 test data types are possible to create the perfect test data management solution for your organization.

"Making the simple complicated is commonplace; Making the complicated awesomely simple, that's creativity."

Charles Mingus.

Biography

Edwin is a Test Manager and Test Advisor at Yacht, specializing in test data management. He has 14 years testing experience of which two were spent specializing in test data management. Edwin is ISTQB full advanced certified. He is a speaker at well-known international and national events like Eurostar, SQC (Iqnite), Testnet and Dutch National Testing Day.

Edwin helps to organize test data management in many large, multi-national organizations for a variety of important programs and projects.

Testing as the conscience of a project

by Ivan Ericsson

The last 20 years have seen testing slowly progress from an inconsistent function made up of organisat-ional waifs and strays towards a respected activity undertaken by qualified and experienced professionals. However, this should not be the summit of our achievement. This article will argue that testing and quality assurance professionals need to strive to be perceived as more than just another organisational function and play their part in delivering organisational IT goals. We need to operate as the conscience of a project, asking the difficult questions, and using our unique perspective to drive informed business and IT decisions.

We will look to answer the following questions:

- What is a project conscience?
- Why is the testing function so well positioned to provide this?
- Whose conscience are we providing?
- Why is this required?
- How to do it
- What is an 'agile' conscience

What is a project conscience?

If a conscience is defined as "a person's sense of right or wrong", how is this applicable to a project?

In the days of segmented changed lifecycles and project management structures with sectioned responsibilities, it is too easy for any one project function to fulfil their own remit and look no further. Testers have traditionally born the brunt of this approach and found defects that other people on the project were either not looking for or were assuming were the responsibility of "someone else".

This has developed in project testers a keen sense of right and wrong, and we naturally feel a responsibility to probe those areas of risk which may fall between the cracks that we often only recognizse the existence of, when applications are put into production and fail to fully deliver the anticipated business function. But it must be realizsed that *a conscience is seldom a popular voice on a project, so it is seldom easy to fulfil this function.*

The risk's the thing, wherein I'll catch the conscience of a king...?

Most key decision makers within organizations will acknowledge the importance of testing – but how many really value the information with which testing provides them and use it in making their decisions? Sadly, not many.

This may be because the message has been confused – of all of the data produced by a test effort, it can be hard to disseminate, especially if this dissemination is not done for them. In order for a test function to effectively operate as an information provider, this message needs to be clear and structured.

A conscience that is not easy to understand can be easily ignored.

In order to be understood the message from testing needs to be clear and easy to understand in the terms of your intended audience. It needs to connect achieved quality with business and project risk. Talking in terms of business functions, which may not work, is the most effective way to be heard – especially if it can be related in terms of potential business or reputation lost. Talking in terms of this risk is the best way to become an audible conscience.

Standing on the shoulders of giants?

A conscience is often portrayed as a figure standing on a shoulder whispering into an ear – and indeed this is often the role played by testing. It is important, however, to ensure that the correct shoulder has been selected. There are often people, who are very happy to listen to talk of delivered quality and perceived risk, but these are not necessarily the right people with whom to communicate. The shoulder needs to be that of someone with influence to do something about the information which is presented. Ideally, this means the business – those who will feel the most pain in the failure of an application to fulfil its purpose.

It is rare (but not unheard of) for an organisation to have a test or quality manager at a high enough position to really influence decisions directly, but then *it is not necessary to be a giant in order to stand on their shoulders*, and often it is better not to be. Every tester should be able to communicate their work and pro-gress in terms of achieved quality and business risk, and often it is the voice of the people who are actually exercising the code which has the biggest impact. Interestingly, we are seeing an increase in active business interaction within projects – opening up a very good channel between the test function and the ultimate sponsor – so increasingly the giant seems to be stooping and listening.

Of what does a conscience talk?

In order to be understood and accepted, the conscience often has to talk in terms or ways that are unex-pected. The basic message has to relate to aspects of the system which may be at risk (and therefore not deliver benefit) if a decision to implement a given solution is taken. Indeed this is the message which is most expected (but rarely well delivered), and the struggle is to overcome

message fatigue by presenting this information in new and innovative ways. This may be by predicting revenue loss, or additional support costs, or talking in terms of damaged reputation and loss of market share.

The conscience will often have to speak in different terms to different stakeholders, to the business in terms of revenue loss (or failing to deliver expected return on opportunities), to project managers in terms of project failure, over-runs and additional cost, to the C level in terms of IT failure to support transition and business advantage.

A conscience must be free enough to talk of those things which easily slip through the gaps, because they are difficult or little understood, or unseen, such as:

- Non-functional quality attributes. Risks around security, resilience, performance or maintainability.
- Areas that have not been tested, due to time constraints, prioritisation, etc.
- Deficiencies in quality assurance processes that may mask finding out the true quality of a solution
- Standard product-based defects (which we are told don't exist) rather than project defects.

Any of these could damage the value of a delivered application, and they are often understood least by those who make the key decisions. Risk may also be present in the very way of working, be it a third-party development organisation, which is being very protective of progress and quality, or a new development model which is facing implementation difficulties.

There are hidden and subtle risks associated with every change, and it is of these that a conscience must talk.

Finally, the conscience should talk in a voice which is fit for its purpose and which will meet its objectives. Good testers are *pessimistic optimists*; they can see the finish line and phrase their concerns in terms of removing impediments in order to get there. It is all too easy for a tester's message to be doom-laden, and this is often why there is a natural reluctance to have a traditional quality voice at key decision points. For the conscience to work, it has to temper its message, to be Machiavellian in the timing and content of its communication. The conscience has a natural *risk filter* and needs to be able to *select the right message, for the right ear, at the right time*.

How to feed the conscience?

The primary feed for the conscience is the structured test effort, this will usually reveal a large percentage of the risks – and a well structured and delivered representation of this data and the resultant risks is essential. Central to this is the use of metrics, using complex data to represent complex trends or risks in a simple way is a critical challenge – but it is vital that wherever possible a conscience has the data at hand to back up what it claims. Otherwise the conscience will be questioned, and this seriously undermines the ability to perform the function.

Exploratory testing is a very good way of identifying areas of risk that would not necessarily be found through more traditional testing efforts. It is a good additional activity, which will often raise interesting questions of the solution through its focus on learning, experience and creativity. A key part of exploratory testing is ad hoc testing, possibly the least formal of all testing types and one specifically designed to exercise the improvisation skills of the testers, to force them off the common path into the murky undergrowth in search of areas of risk that would not have been otherwise found. It is always a good practice to plan some time for this kind of testing activity. A good tester is finely attuned to areas of risk, and like a shark sensing blood will be able to focus, often from a distance, on those areas of weakness.

Another way to probe the project risk boundaries and provide the conscience with ammunition is to lead risk reviews – and in doing so force groups to use their expertise to uncover unconsidered aspects of the solu-tion. People such as architects, data analysts and compliance and conformity specialists are ideal target audiences for this kind of activity.

A project model that has already adapted to a lot of these perspectives is Agile. Agile has (in theory) a shared conscience, the entire team is a living conscience, and we have a lot to learn from it in this respect.

An Agile conscience

The move of testing from a driven function to a driving function has already started at pace within the various agile development models which are increasingly being employed. All members of an agile team are held responsible for the quality of the delivered solution. There is a continual and joint focus on the solution being built and its appropriateness for the end goal. The open and regular communication model creates a healthy environment for the discussion of risk, and the business sponsor has, in effect, a large collective conscience on its shoulders.

That said, there are some weaknesses even in agile (the practice of, not the theory), which should be con-sidered.

Firstly, the focus on achieving business satisfaction can sometimes direct the team's attention towards removing risks, which are obvious to the end user (look and feel, basic functionality), whilst paying less attention to those areas with which the end user would be less conversant. To this end the agile team needs to include a range of skills and perspectives, and they all need to be encouraged to contribute from their knowledge base and experience to the collective role as conscience.

Secondly, although the number of agile projects is increasing, the number of people who would consider themselves 'agile' is still quite small. Getting people out of a waterfall mindset, remoulding their natural reac-tions and reshaping their risk instincts, is more difficult than just putting some post-it stickers on a wall or having daily stand-ups.

Despite these concerns, agile projects do have a natural orientation towards project conscience as a basic operating principle and have shown some of the ways forward for testing as a key function.

Summary

Testers do have a unique perspective, and with this comes a unique responsibility to help organizations make the right decisions. They do this by operating as a project conscience, by informing the key decision makers in clear and concise terms of risks to business or reputation.

To succeed in this, we need to select the correct shoulder and the correct information – we need to shine a flashlight into the dark places of a project and speak of what we see.

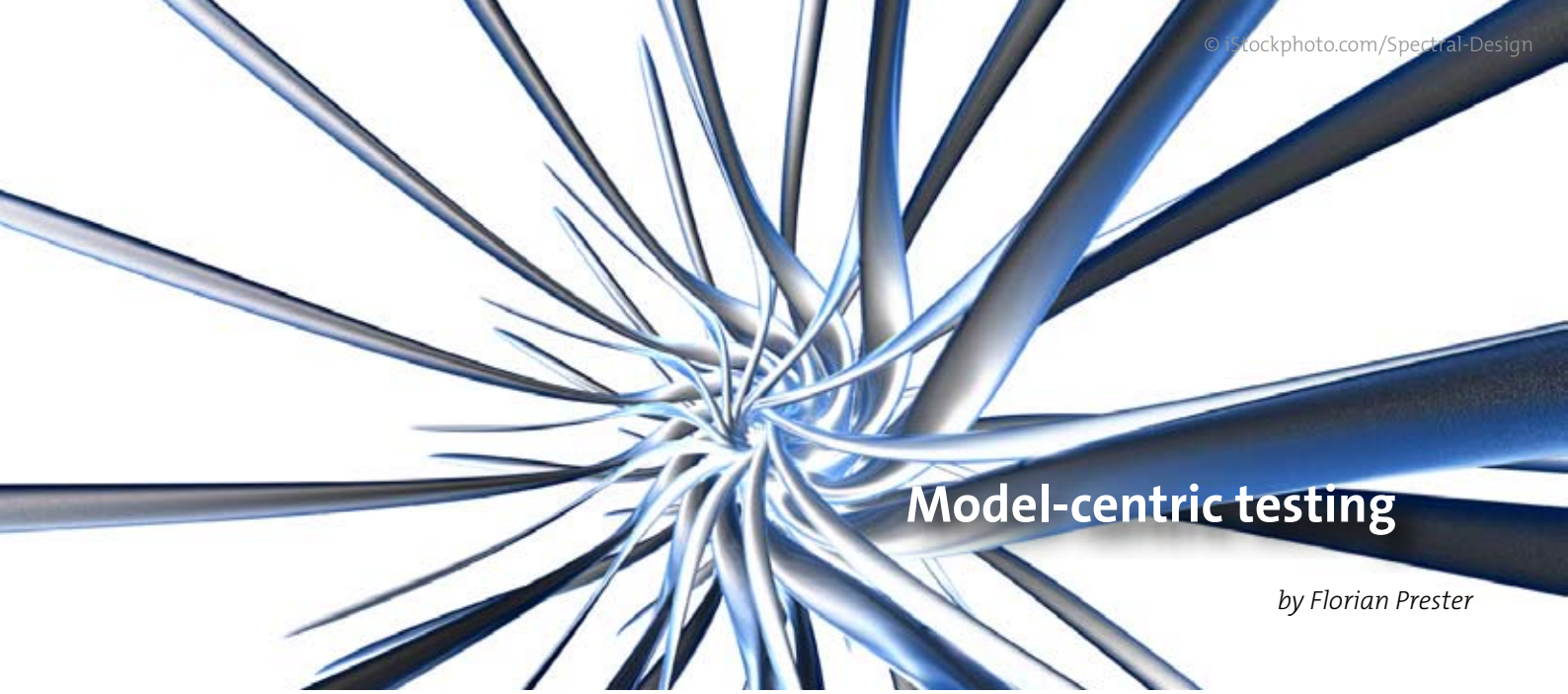
Increasingly, we are observing an active business involvement in projects of IT change – seen perhaps in the agile model more than anywhere else. This is good for organisations, good for projects and provides a natural and interested audience for the communication of risk. Testers should be exploiting this and actively work as a project conscience.

The Agile approach of promoting testing from 'just another function' to a central remit of the working model has opened a door for us and given us a view of where we, as the testing profession, could be. To get there, we need to be able to support key project decisions from an informed strategic viewpoint, and to do this, we need to take a close look at our working models and relationships.

Testing needs to step onto the shoulder, and have something to say when we get there.

Biography

Ivan has been a Test Manager for 15 years in a range of sectors and working for organisations throughout Europe. He is a Director with SQS and, as well as currently leading a strategic project in Sweden, he is re-sponsible for the SQS special interest groups and service innovation. A regular presenter at test confer-ences, he is particularly inter-ested in test management and test process improvement.



Model-centric testing

by Florian Prester

Model-centric testing (.mzT[1]) is an efficient method to design and specify (software) tests. By using .mzT it is possible to achieve a systematic approach and improve the completeness of the test coverage. .mzT is based on a model-based test design, extended by test management information and the tester's mindset. .mzT can be used for every kind of software and system test. It serves all relevant parts of software development, beginning with component view, which is focused on complete test coverage, and ending at system view, which in respect to the number of possible test cases primarily focuses on a systematic reduction of test cases.

In this article, the methodology of .mzT is introduced, and for better understanding a scientific project and a prototype are explained.

Introduction

Model-based testing has been known for years. It is mostly based on deriving test cases out of development information. Over the last years, many promising ideas were developed and techniques as well as tool chains were established to serve that idea. All of those concepts have one drawback in common – using development information alone only provides verification but not validation. A development model of a software component can be used to generate test cases which will cover c0 to c3 coverage metrics, but it cannot be used to generate test cases which are "outside" of the model.

Both techniques (verification and validation) strive for a complete test system, instead of describing single test cases and generating the single test cases out of it automatically by using a test case generator.

.mzT – model-centric testing

Model-centric testing (.mzT) is a logical further development of model-based testing (MBT) and therefore a method for the visualization and systematization of test designs and test plans. Whereas with MBT, implementation is verified based on models from the system definition or the system design, .mzT goes one step further and adds system usage and test management aspects to the usage/behavior models (Fig. 1).

Thus .mzT focuses primarily on the validation: to load the system similar to the user workflow (usage) and to add exceptional situations/worst cases (tester's mindset), while including the procedures and objectives of the model-based approach. This ensures completeness of the test and therefore high test quality and test coverage, particularly with regard to the validation objectives. The integration of test management information into the model (such as priorities) (Fig. 1) and the automatic generation of test cases from the model, that takes into account the test management information and test requirements, also allows a relevant and test target oriented reduction of the test cases ("best test cases"), thus ensuring an economically viable test process. A schematic diagram of the .mzT process is given in Fig. 2.

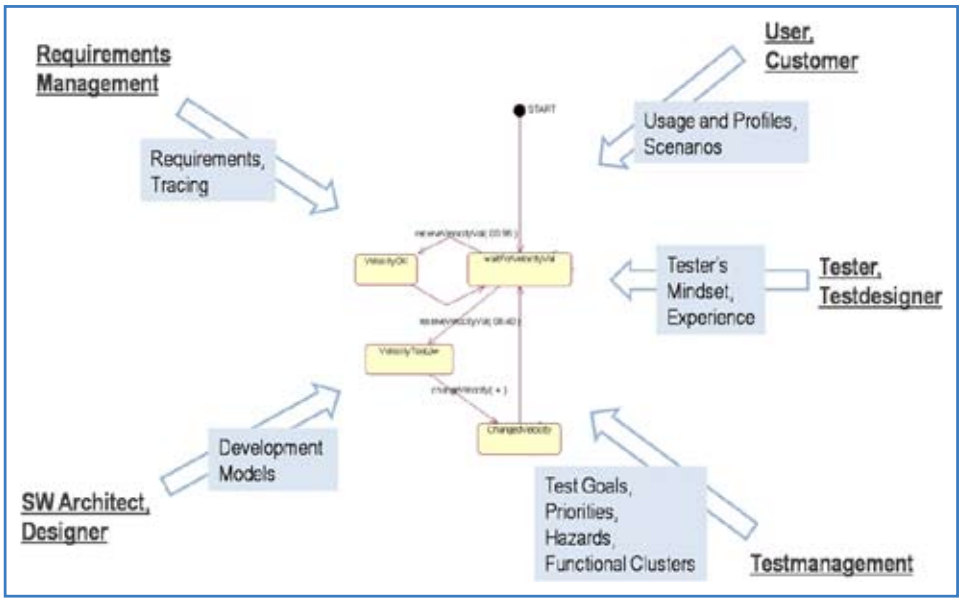


Fig. 1: Model-centric testing (.mzT): In addition to possible input from development models, test modelling is based on user scenarios, test management information and necessary test cases from the viewpoint of the test (exceptional scenarios, error scenarios).

The advantages of .mzT as opposed to a "traditional" document or script-based approach are:

- **Systematic** design and generation of test cases, providing

1 .mzT = modellzentriertes Testen (German), English: model-centric testing

controllable completeness in terms of coverage and in the reduction of test cases by test management criteria (for example, project risks and knowledge of path coverage in the case of test case reduction).

- **Visualization**: both developers and testers make a model of the system at least in their heads. Using .mzT both work on the same visual model, typically based on UML (Unified Modelling Language). This ensures better coordination of the test suites including coordination between stakeholders, requirements managers and developers.
- **Reuse** of information: Avoiding redundancy. The individual test runs no longer have to be created manually and maintained individually with great effort.
- **Modularity** in the models and the possibility of using different levels of abstraction by means of hierarchies in the test design leads to a reduced complexity of the design.
- **Automation** of generating test cases and of the connection to test management and test execution tools provides economic efficiency and avoidance of errors. (Errors typically occur when test attributes, such as requirements, are held redundantly in the test design and in the test management tool.)

test report, .mzT and .getmore provide an automated tool chain without gaps that places the .mzT model at the centre of the test activities.

Research project TestNGMed

TestNGMed is a specialized sector solution of an automated test bed for Medical IT, in which .mzT is applied with the aim of high-quality and economically efficient validation of medical engineering in a clinical environment, based on domain and test standards (see Fig. 3). The tool chain implemented for this project consists of a modelling tool (Enterprise Architect, Sparx Systems), the .getmore test case generator (sepp.med gmbh) and the execution environment (TTworkbench by Testing Technologies).

In the domain of medical engineering, the protocol standard HL7 is used as the communication and interaction protocol, and IHE as the standardized definition of medical processes in TestNGMed. Manufacturers orient themselves towards IHE as a reference, but it often deviates from practical operation in a clinical environment quite considerably and in very specific ways. Here the efficient adaptability and expandability of the test designs with the .mzT methodology comes into its own. IHE scenarios form the framework of the test design and, based on the special clinical processes and on the tester's mindset, are expanded to become high-quality validation suites adapted to the specific system.

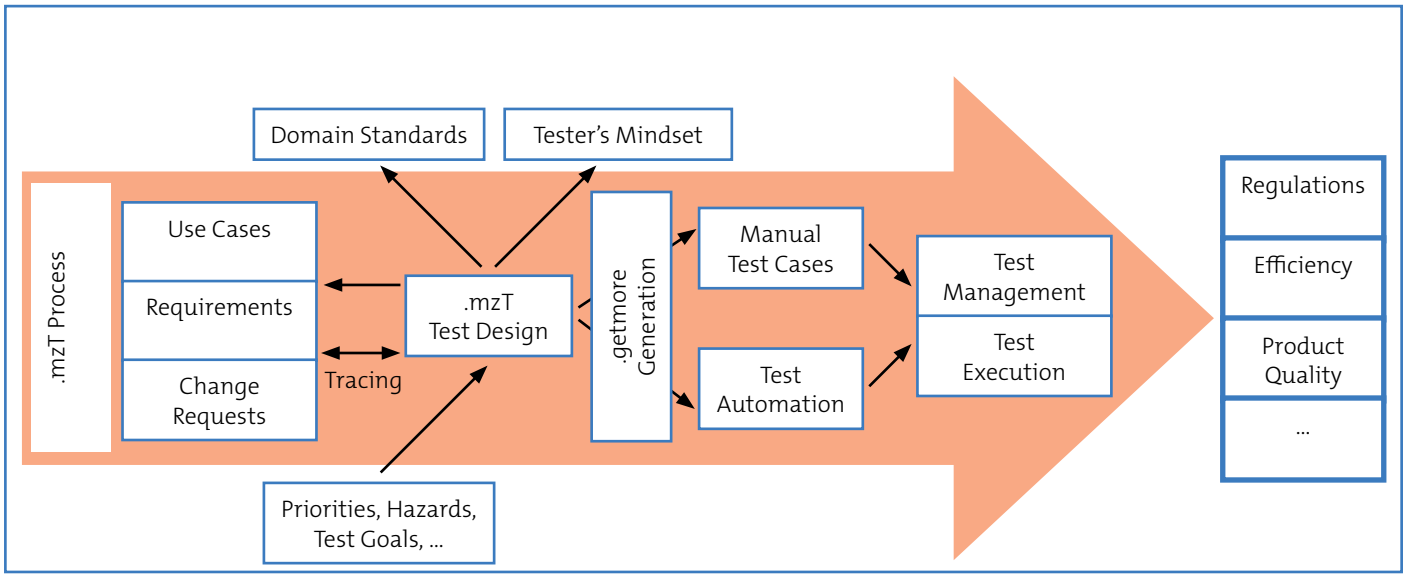


Fig. 2: .mzT integration into the test and development process

Because .mzT is independent of modelling, test management and test execution tools, it is possible to systematize the test design without having to modify existing test systems or processes.

The .mzT test design is integrated consistently into the software development process. This allows the automation of process steps, resulting in higher process quality and efficiency. Also tracing between requirement management, development and testing is provided. The inputs to the test design are not only product demands, such as use cases, but also requirements, change requests and the tester's mindset. In addition, usable and applicable domain standards (e.g. AUTOSAR), the experience of experts and users of the system are also included.

The test model is the central repository of the test process and contains not only the test case information, but also attributes that are relevant to the test management or the project management (priorities, functional clusters, endangerment relevance, costs, etc.). This permits extensive planning and control of the test process from the test model and considerably improved adaptability of the test design to the requirements of the test process. For instance, it is possible to generate those test cases from the .mzT model that correspond to the test scope and the planned priorities for regression testing. From the requirement to the

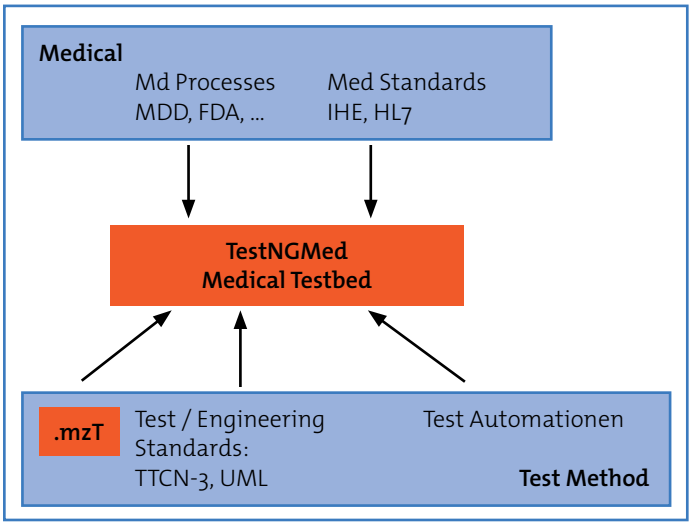


Fig. 3: Established standards as the basis of the test bed

The integration of the test management aspects into the test models permits a high level of traceability between manufacturer processes and hospital processes and integration into the development process. This is of decisive importance in the regulatory environment of medical engineering projects, for example, the

German Medical Products Act (MPG) and the regulation requirements of the U.S. Food and Drug Administration (FDA).

Moreover, largely manual test processes are applied at the interfaces between manufacturers and the hospital, so that the structured, analytical and justifiable reduction of the test cases based on test management information, such as endangerment relevance, is essential here.

TestNGMed therefore addresses, above all, the topics of conformity with standards and interoperability. With the TestNGMed scenarios, it is possible to test both the IT infrastructure as a whole and the individual systems, in the latter case, with TestNGMed acting as a simulator of the environment. The complete test process with TestNGMed is shown in Fig. 4.

The test methodology uses the engineering standard UML for modelling the test suites and makes use of test automation. The test automation is implemented using the test standard TTCN-3 (Test and Test Control Notation). The test environment TTworkbench was expanded to include the adapters for the HL7 protocol. The test design is executed completely in the model, and the executable TTCN-3 test suite is then generated with the .getmore generation tool. Because the test bed is based on the HL7 protocol and not on the user interface, as it normally would be, the maintenance costs usually associated with user interface based tests can be drastically reduced.

TestNGMed was developed to maturity for use in a clinical laboratory process environment as part of a national and EU-wide subsidized research project. The project partners are TU Berlin (Prof. Dr. Ina Schieferdecker) for the test automation methodology and TTCN-3, Applied Biosignals as the product supplier (Polybench, a medical data flow analyzer) and validation partner for the test of the laboratory workflows. sepp.med provides the expertise on .mzT in its use in the medical domain and the test case generation. The adaptations of the domain-independent solution with respect to the requirements of medical processes are based on years of extensive project experience, for example in radiology (scanners and hospital IT) and oncology (particle therapy), resulting in a regulatory-compliant test bed, that is both efficient and effective as well as very usable.

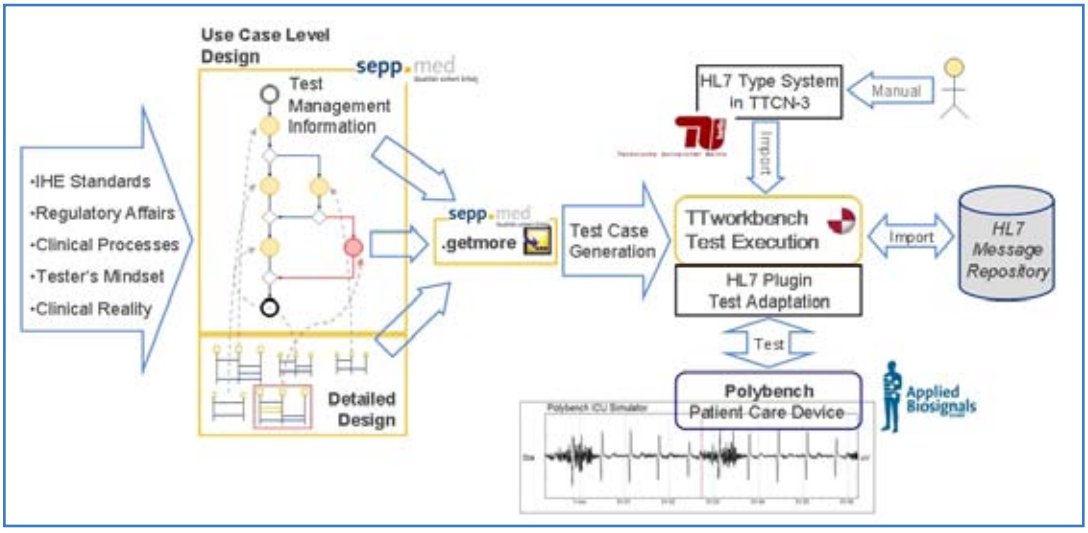


Fig. 4: TestNGMed overall system in the ProInno project

Initial industrial projects are currently being planned for the deployment of TestNGMed in future projects with medical engineering manufacturers and in the integration of clinical IT. For this purpose, the HL7 implementations and the IHE profiles supported are continually being expanded as part of these projects and as part of funded projects, and the test bed is being optimized based on deployment experience and the validation results. The long-term goal and vision of TestNGMed is to provide a standardized test suite that is to be developed, for example as part of product certification, into an industry standard for medical devices.

Prototype TestNGMost

In a cooperation project between Ruetz System Solutions GmbH, Testing Technologies IST GmbH and sepp.med gmbh, the extension of the TTCN-3-based test and simulation system TTsuite MOST with the model-centric test design approach was evaluated. MOST (Media Oriented Systems Transport) is the leading network standard for car infotainment and is used for the transmission of audio, video, speech and data signals on fiber-optic waveguides.

Over the MOST bus, it is for example possible to operate an entertainment system remotely and to replay the audio data on a player connected to the MOST bus (Fig. 5). The objective of this project is to optimally test the remote operation possibilities while maximizing test efficiency: creating a complete test design and reducing the generated test cases in accordance to process, quality and release goals.

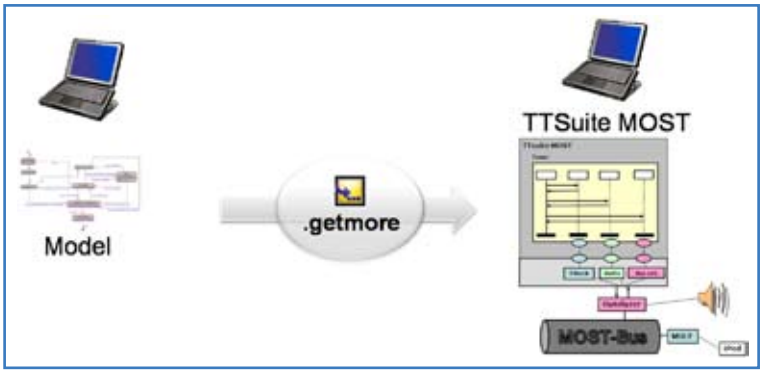


Fig. 5: Schematic test setup TestNGMost

The functions to be tested include *play, pause, forward, scan, shuffle*, and *list*, which can be executed in any order. Based on the description of all available functions, a test model (.mzT) was created using the model-centric testing methodology. One central point that embodies the difference between the test model and pure development models is modelling of the "tester's mindset" and the test management information in the model. Using the test case generator .getmore, test cases are systematically generated from this test model. The coverage and combination logic can be controlled using strategies, priorities and flow conditions.

The generated test cases are available in TTCN-3 format and can be used directly for execution in the TTsuite MOST test automation framework. In addition to test case generation, the model is used as a basis for communication between all people involved in the development process.

Unlike with intuitive, manual test case preparation, this approach makes it possible to systematically identify all test cases and to cut down the number of test cases in a systematic way, if necessary.

The basis of the test setup are the TTsuite MOST with libraries for

the MOST connection of the test system, the hierarchical .mzT test model and the infotainment system of a real vehicle. After the test cases have been generated from the test model using .getmore, the test cases are compiled with TTsuite MOST and are immediately available for execution. Through an interface (Optolyzer), that establishes the connection between the PC and the MOST bus, the commands are sent to the iPod over the MOST bus. The music data from the iPod is transmitted over the MOST bus to the appropriate output device (loudspeakers).

Conclusion

Model-centric testing shows that MBT is nothing theoretical, but can be used successfully in practice. In the course of many industrial projects, .mzT grew from a small idea to a real methodology including guidelines, training and tooling. .mzT helps the test managers to organize their processes according to existing tool chains and extend them by MBT/.mzT.

A young and growing community can be found under www.modellzentriertestesten.de, an exchange platform for model-based testers. Feel free to join experts and chat about MBT/.mzT.

Biography

From 1998 to 2004 Florian Prester studied Computational Engineering at the University of Erlangen-Nuremberg. After his degree he worked as research assistant at the Regionales RechenZentrum Erlangen-Nuremberg (RRZE) with special interest on network security and abstraction mechanism. Since 2007 he has been working for the sepp.med GmbH, where since 2009 he is CEO responsible for the automotive, automation & systems industries. His interests are test conception, modelling and test improvement.

Managing the Testing Process

Training Course by Rex Black

June 09 to 11, 2010 in Bad Homburg

limited places

Test managers must take a potentially infinite job—testing a computer system—and accomplish it within tight time and resource restraints. It's a tall order, but successful test managers have found proven ways to handle the challenges.

This course will give attendees the tools they need to succeed as test managers. We'll look at quality risk analysis, test estimation, and test planning. We'll discuss developing high-quality test systems—test cases, test data, test tools, even automated test systems—that improve over time. We'll talk about tracking bugs and test cases. We'll discuss ways to derive and present metrics, charts, and graphs from the test results.

We'll also cover the human side of test management. We'll look at ways to measure and manage the skills testers need. We'll discuss hiring testers. We'll talk about education and certification for testers. We'll examine some ways to motivate and reward testers—and some ways not to! We'll cover working effectively within the project organization, which is especially challenging when you're the bearer of bad news.

We'll also look at the context of testing. We'll discuss system development lifecycles and how they affect testing. We'll cover testing as an investment. We'll finish up by discussing test labs, test environments, and hardware issues.

The materials presented in the course follow Rex Black's book, Managing the Testing Process, which is the distillation of over two decades of software, hardware, and systems experience.

www.testingexperience.com/knowledge_transfer.html

1200,- €
(plus VAT)

te testing experience
Knowledge Transfer

Masthead

EDITOR
Díaz & Hilterscheid
Unternehmensberatung GmbH
Kurfürstendamm 179
10707 Berlin, Germany

Phone: +49 (0)30 74 76 28-0 Fax: +49 (0)30 74 76 28-99 E-Mail: info@diazhilterscheid.de

Díaz & Hilterscheid is a member of "Verband der Zeitschriftenverleger Berlin-Brandenburg e.V."

EDITORIAL
José Díaz

LAYOUT & DESIGN
Katrin Schülke

WEBSITE
www.testingexperience.com

ARTICLES & AUTHORS
editorial@testingexperience.com

350.000 readers

ADVERTISEMENTS
sales@testingexperience.com

SUBSCRIBE
www.testingexperience.com/subscribe.php

PRICE
online version: free of charge -> www.testingexperience.com
print version: 8,00 € (plus shipping) -> www.testingexperience-shop.com

ISSN 1866-5705

Index Of Advertisers

Testing & Finance 2010

The Conference for Testing & Finance Professionals

June 7th and 8th, 2010

in Bad Homburg (near Frankfurt a. M., Germany)

More information and registration at www.testingfinance.com

Keynotes by

Nitin Bhargava

Barclays Bank, UK

BJ Rollison

Microsoft Corp

Gernot Nolte

Fiducia

Prof. Dr. Hermann Schulte-Mattler

Fachhochschule Dortmund

Rex Black

RBCS

Sponsors

CS Consulting AG

Exhibitors

BSM · CaseMaker · hp · iSQI · logica · Microsoft · T-Systems

Supporting Organisations

VOB SERVICE · testing experience · Díaz Hilterscheid

Testing & Finance 2010 - The Conference for Testing & Finance Professionals

June 7th and 8th, 2010 in Bad Homburg

Day 1	Track 1	Track 2	Track 3	Track 4 - Finance
9:10	Opening Speech, José Díaz			
9:15	Key Note - Nitin Bhargava - Barclays Bank, UK			
10:15	Break			
10:20	"Exploratory testing explained" Alon Linetzki	"Allheilmittel Test-Automatisierung?" Thomas Lingenfelder	"Managing a Lean test process; is testing waste?" Chris Schotanus	"Zweite MaRisk Novelle - Auswirkungen auf die Banken IT" Matthias Korsch
11:10	Coffee Break			
11:30	"Test Governance structure" Holger Bonde & Thomas Axen	"TM2010 - Doppelpass zum Erfolg Unsere Erfahrungen mit TPI®, ISTQB®-Standards und deren Anwendung" Tom Schütz & Dr. Peter Hübner	"SCRUM & Testing; Back to the Future" Erik van Veenendaal	"Post Merger Integration einer Hypothekenbank" Volker Sobieroy & Dr. Jörn Rank & Patrick Esperstedt
12:25	Key Note "How we test at Microsoft" - BJ Rollison - Microsoft Corp.			
13:25	Lunch			
14:40	"Random Testing in a Trading System" Noah Höjeberg	"Testautomatisierung durch MBT" Florian Prester	"Ausgelagerte Testautomatisierung in einem agilen Softwareprojekt" Michael Böll	"Aktuelle Änderungen im Aufsichtsrecht" Dr. Karl Dürselen
15:35	"Governing Testing of Systems of Systems" Bernard Homès	"It won't Work, it Won't Work. It Worked" Jamie Dobson	"Roles and responsibilities of the tester in a Scrum team" Danny Kovatch	"Umsetzung neuer aufsichtsrechtlicher Anforderungen mit BAIS Erfahrungsbericht eines BAIS Anwenders" DEXIA Kommunalbank Deutschland AG
16:25	Coffee Break			
16:40	"Start and Exit Criteria for Testing" Hans Schaefer	"Knowledge is Power: Do you know where your quality is tonight?" Alexandra Imrie & Hans-Joachim Brede	"BAIS – Optimale Transparenz im Meldewesen" Christian Karner "Konzeption – Entwicklung – Testen Überlegungen zu einer zukunftsträchtigen IT-Architektur für das Meldewesen eines Kreditinstituts" Hosam Elsayed	"Erfahrungsbericht Testautomation im Landesbanken Rating" Volker Töwe
17:35	Key Note "Wie manage ich die Softwareentwicklung einer großen Rechenzentrale mit rund 200 Projekten?" Gernot Nolte - Fiducia			
18:40	Social Event Dinner + Theatre and Chill out			

Day 2	Track 1	Track 2	Track 3	Track 4 - Finance
9:10	Key Note "Das Basel II Puzzle: angemessene Eigenkapitalausstattung auf dem Prüfstand", Prof. Dr. Schulte-Mattler - Fachhochschule Dortmund			
10:15	"Test Process Improvement: Critical Success Factors" Graham Bath	"Flexible testing environments in the cloud" Jonas Hermansson	"Agile approach for testing in a high risk legacy environment" Jan Rodenburg	"Verbriefung und Refinanzierung – Plattformstrategie in der Erstellung und Nutzung von Standardsoftware" Klaus Leitner
11:05	Coffee Break			
11:25	"TMMi" Geoff Thompson	"Risk based test planning" Dr. Hartwig Schwier	"IT-Governance, notwendig oder überdimensioniert? Wie ich mich der Herausforderung einer effizienten IT stellen kann." Arjan Brands & Oliver Rupnow "Effiziente Qualitätssicherung über risikobewertendes Requirement Management" Thomas Köppner	"Neue Europäische Bankenaufsicht - Auswirkungen auf das deutsche Meldewesen " Nicolas Jost
12:20	Key Note "Satisfying Test Stakeholders ", Rex Black			
13:20	Lunch			
14:30	"Improving quality and saving costs using static analysis: a case study" Dr. Mike Bartley	"Testprozess bei der Volkswagen Financial Service AG - Keine Theorie sondern gelebte Praxis" Ingolf Gäbel & Yvonne Sternberg	"Qualität von Anfang an – Anforderungsbasiertes Testen mit Microsoft Visual Studio Test Professional" Matthias Zieger "Requirements Engineering and Certification" Benjamin Timmermanns	"Ertragsorientiertes Liquiditätsrisikomanagement - Erhöhung der Erträge durch bessere Liquiditätsrisikoanalyse in Zeiten rückläufiger Ergebnisse" Prof. Dr. Stafan Zeranski
15:20	Coffee Break			
15:25	"TMMi - Model Development and practical expierences" Matthias Rasking	"SOA testen - der inkrementelle fachliche Integrationstest" Ronald Grindle	CS Consulting AG	"Pros und Cons zur Einführung einer Leverage-Ratio als Ergänzung der risikosensitiven Basel II-Normen" Dr. Uwe Gaumert
16:20	Closing Session, José Díaz			

Training with a View

Díaz Hilterscheid

also onsite training worldwide in German, English, Spanish, French at
http://training.diazhilterscheid.com/
training@diazhilterscheid.com

"A casual lecture style by Mr. Lieblang, and dry, incisive comments in-between. My attention was correspondingly high. With this preparation the exam was easy."
Mirko Gossler, T-Systems Multimedia Solutions GmbH

"Thanks for the entertaining introduction to a complex topic and the thorough preparation for the certification. Who would have thought that ravens and cockroaches can be so important in software testing"
Gerlinde Suling, Siemens AG

- subject to modifications -

15.03.10-17.03.10	Certified Tester Foundation Level - Kompaktkurs	Berlin
22.03.10-26.03.10	Certified Tester Advanced Level - TEST ANALYST	Düsseldorf
22.03.10-26.03.10	Certified Tester Advanced Level - TESTMANAGER	Berlin
12.04.10-15.04.10	Certified Tester Foundation Level	Berlin
19.04.10-21.04.10	Certified Tester Foundation Level - Kompaktkurs	Hamburg
21.04.10-23.04.10	Certified Professional for Requirements Engineering - Foundation Level	Berlin
28.04.10-30.04.10	Certified Tester Foundation Level - Kompaktkurs	Düsseldorf
03.05.10-07.05.10	Certified Tester Advanced Level - TESTMANAGER	Frankfurt am Main
03.05.10-07.05.10	Certified Tester - TECHNICAL TEST ANALYST	Berlin
10.05.10-12.05.10	Certified Tester Foundation Level - Kompaktkurs	Berlin
17.05.10-21.05.10	Certified Tester Advanced Level - TEST ANALYST	Berlin
07.06.10-09.06.10	Certified Tester Foundation Level - Kompaktkurs	Hannover
09.06.10-11.06.10	Certified Professional for Requirements Engineering - Foundation Level	Berlin
14.06.10-18.06.10	Certified Tester Advanced Level - TESTMANAGER	Berlin
21.06.10-24.06.10	Certified Tester Foundation Level	Dresden

Kurfürstendamm, Berlin © Katrin Schülke

ACCREDITED ISTQB Certified Tester Advanced Level TRAINING PROVIDER