17

March 2012

testing experience

The Magazine for Professional Testers

printed in Germany

print version 8,00 €

free digital version

www.testingexperience.com

ISSN 1866-5705

Model-Based Testing

2013 software quality days

15.-17. Januar 2013

in Wien

Call for Papers

Software Quality Days

Der jährliche Kongress für Software Qualität & Testing

Reichen Sie jetzt Ihren Fachbeitrag für 2013 ein!

www.software-quality-days.com

Software Quality Days sind eine der größten Konferenzen zum Thema Software Qualität in Europa und ein Fixpunkt für jeden Interessierten in den Bereichen Software-Qualität, Testen und qualitätsorientierte System- und Software-Entwicklung.

Termin:	15. bis 17. Januar 2013 in Wien
Einreichfrist:	ab sofort bis 20. Mai 2012
Einreich-optionen:	Workshop/Tutorial, Praxisvortrag, Praxisvortrag erweitert, Experten-Talk, Wissenschaftlicher Vortrag

Qualität - Investition in die Zukunft

Themen für Vorträge & Tutorials:

Requirements, Modellierung & Architektur

Testen & Testautomatisierung

Agile Methoden, Prozesse & Qualität

Embedded Systems & Mechatronik

Integration & Deployment

Usability & Security

Veranstalter: **Software Quality Lab GmbH**

[T] +43 732 890072-0 [E] einreichung@software-quality-lab.com

Editorial

Dear readers,

If you live in Europe I hope you enjoyed the cold front. It was nice to have few days for ice skating. However, to tell the truth I'm also happy that the weather changed again and we can look forward to spring.

I love how the seasons change. Each season has more or less its own model, even though it is getting more difficult to predict due to the climate change.

Testing can also be based on a model. In this issue, we have some great articles that explain where Model-Based Testing is coming from, where it is today, and where it can be expected to go. There are some tools helping you to test based on models. We have had very few issues of our magazine where so many people were interested in sending us articles. It is amazing to see that the community for MBT is so large. I want to thank all of the authors, supporters and customers for helping us to issue this magazine. I think that you will all love it.

A few days after the magazine will have been published, we will have the Belgium Testing Days. Don't miss it. Have a look at the program at www.belgiumtestingdays.com.

It looks like we really took the right decision to move the Testing & Finance from Frankfurt to London. It will be a great conference. If you are involved in the finance industry, don't miss the European Conference "Testing & Finance". Have a look at www.testingfinance.com

In the next issue of our magazine we will present you the new editorial board. We will have some colleagues worldwide helping me to choose the articles. We will also have a paper submit system to organize it. Our back office is working hard to set it all up. We already did this for Agile Record, our other magazine.

The news that did surprise me quite positively was that the BCS (British Computer Society) will run the certification CAT – Certified Agile Tester. I think that this is the right decision if you consider how good and successful the training is. As far as I know, this is the only certification that has put right what certification haters have all the time been criticizing about other certifications. In CAT, testers will get a personal assessment, a practical exam and a written exam (no multiple choice!). 50% of the training will be exercises. Please read the BCS press release on page 13.

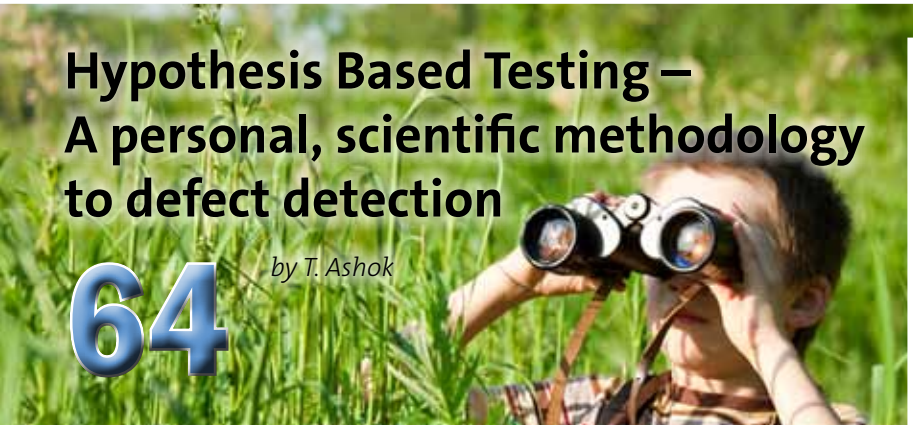
I wish you a pleasant time, that you enjoy the weather models of the next weeks, and also that you learn a lot reading the hands-on articles in this issue.

Best regards

José

testing experience

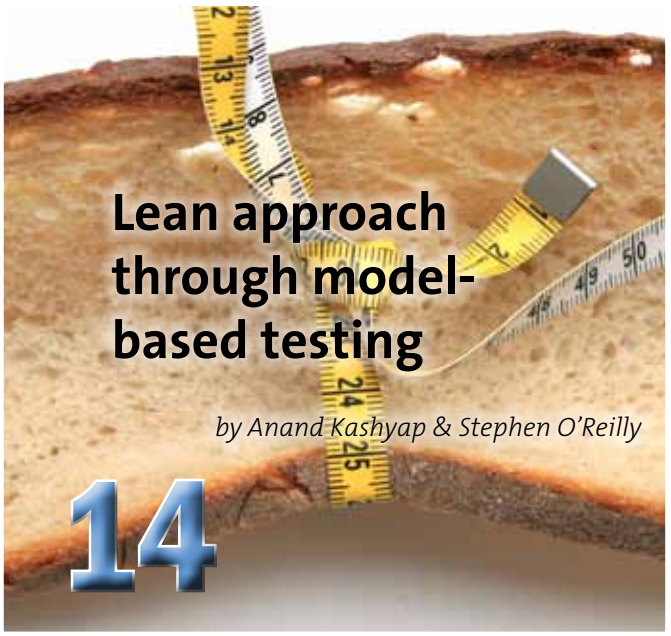
Contents



Hypothesis Based Testing – A personal, scientific methodology to defect detection

by T. Ashok

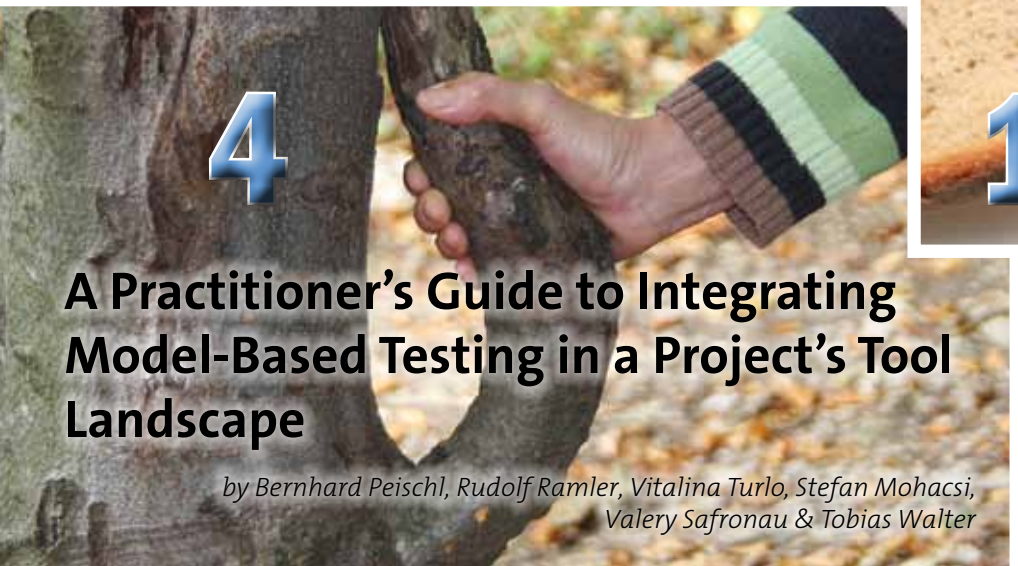
64



Lean approach through model-based testing

by Anand Kashyap & Stephen O'Reilly

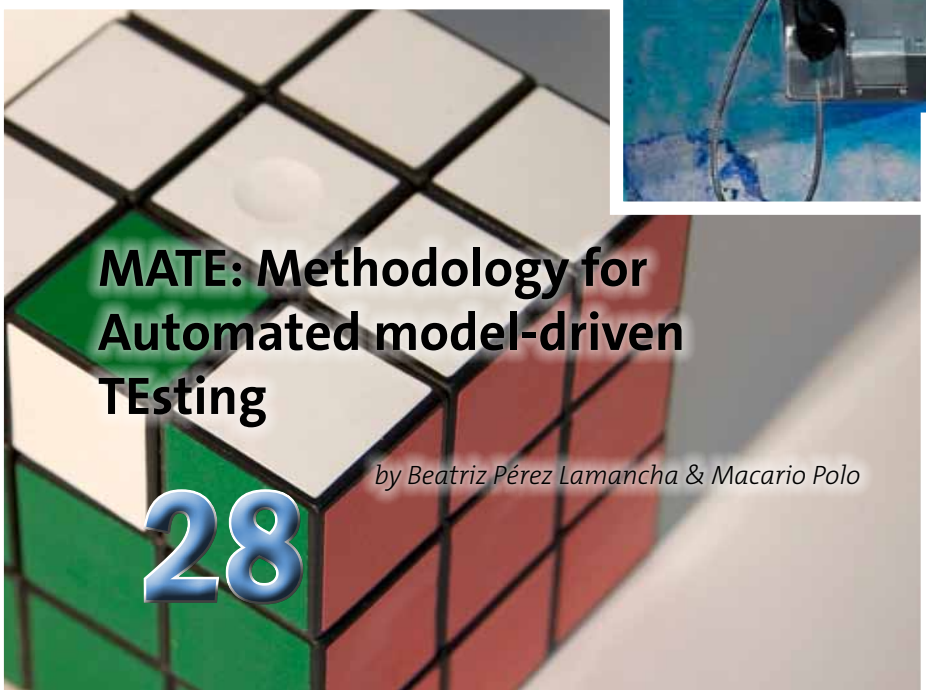
14



4

A Practitioner's Guide to Integrating Model-Based Testing in a Project's Tool Landscape

by Bernhard Peischl, Rudolf Ramler, Vitalina Turlo, Stefan Mohacsi, Valery Safronau & Tobias Walter



A Practitioner's Guide to Integrating Model-Based Testing in a Project's Tool Landscape

by Bernhard Peischl, Rudolf Ramler, Vitalina Turlo, Stefan Mohacsi, Valery Safronau & Tobias Walter

Motivation

Software testing is a knowledge-intensive task that is driven by the information flow generated from the creativity, motivation and cooperation of the involved people. Repetitive mechanical activities, manual steps in automated processes, an unaligned infrastructure and disruptive workflows create painful blockers for this information flow. As some test-related tools have their own proprietary data-storage approach, the resulting isolation makes sharing information increasingly difficult, and tools that are not aligned properly may lead to unneeded complexity and manual test-related activities [Hüttermann 2011]. Hence, the creation of test cases alongside the corresponding test data is increasingly supported by model-based testing (MBT) tools. MBT promises (1) the detection of faults at an early stage in the development cycle and (2) reduces the maintenance effort of test scripts as the test cases are deduced from the model. Mature processes [van Ewijk 2011] and a smooth integration of the underlying tools [Turlo & Safronau 2011] allow test engineers to get rid of mechanical (test-related) activities and focus their time on the development of models for the generation of test cases.

Impacts of an Inadequate Software Testing Infrastructure

The infrastructure for testing has considerable influence on the costs for finding and fixing errant behavior in software. Based on software developer and user surveys, in the US, the annual cost of inadequate infrastructure for software testing is estimated to range from $22.2 to $59.5 billion. Over half of the costs are borne by software users in the form of error avoidance and mitigation activities. The remaining costs are borne by software developers; this reflects the additional testing resources that are consumed due to inadequate testing tools and methods [NIST 2002].

The authors of [Turlo and Safronau 2011] present a well-timed survey (see Figure 1) highlighting that manylots everyday routine activities in automated testing are still performed manually. The results provide empirical proof of inadequate testing infrastructure due to a lack of appropriate integration functionality.

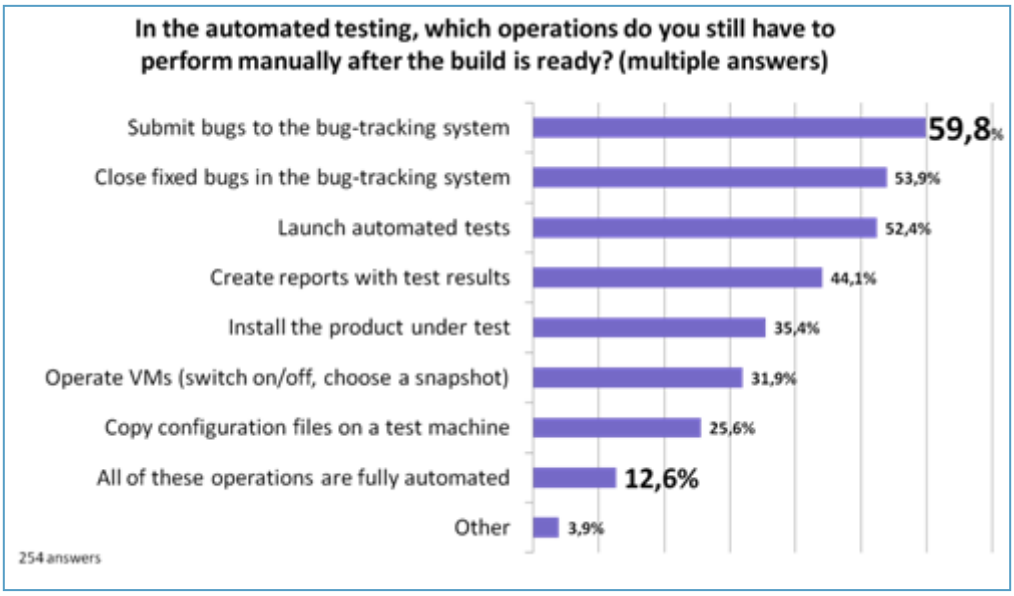


Figure 1: Poorly integrated test infrastructure – top testing activities performed manually during automated testing [Turlo and Safronov 2011].

SWE Guild

The Software Engineering Guild

The online survey on challenges of automated software testing was provided to 11.895 readers of the testing blog hosted at www.habrahabr.ru. From May to June 2011, 299 respondents participated in the survey, offering valuable insights into the level of maturity of automated software testing and tool infrastructure in use. Regarding the job distribution of the respondents, 49.3% are testers, 15% – test leads, 11,9% – heads of QA/testing departments, 23.8% – other, mostly software developers. As for the type of software their organization develop, 65.8% of the participants answered "web" applications, 58.5% – desktop software, 22.1% – mobile applications, and 12.4% – embedded software (the question allowed for multiple answers). Almost 45% of the respondents actively use automated tests in the process carrying out testing and, around 27% use them occasionally, while the remaining 28% – rarely or never.

To evaluate the extent of manual effort expended in automated testing, the participants were asked to answer the following question: "During the automated testing, what are the operations that you still have to perform manually?" This particular question received 247 answers, – 45 respondents chose to not provide a response.

According to the poll, only 12.6% of respondents claim that all the operations related to test execution are performed automatically in their organization, i.e., without human intervention. As Figure 1 shows, the most common tasks that a tester has to carry out manually are submitting and closing bugs in the bug tracking systems (59.8% and 53.9%, respectively), launching automated tests (52.4%) creating reports (44.1%), and installing the product under test (35.4%). As the question allowed for multiple answers, the total percentage exceeds 100 %. These routine menial operations, depending on the number of tests to be executed and the frequency of new builds, might consume as much as 25% of tester work time and result in inefficient use of testing resources[1].

1 One should consider that such potential sources of errors as question wording, question ordering and nonresponse, can introduce error or bias into the findings. We have shown only a portion of survey results in this article. To access the data on tool infrastructure in use, please contact research@appsys.net.

These results along with findings in [NIST 2002] are evidence of the lack of proper cohesion between tools and systems used in software development and testing processes, namely build machines, versioning control systems, virtualization servers, bug tracking systems, and test automation tools. Improving the integration of the testing tools reliefs the test engineer from various manual tasks and thus his/her time and skills can be better allocated, for example, to making model-based testing (including the creation of adequate models) an integral part of the development and test lifecycle.

With the proliferation of test process improvements we have an instrument for systematically setting up a test processes with adequate levels of maturity. For example, leveraging model-based testing requires establishing a certain maturity regarding different aspects of the test process [TE 2011, van Ewijk 2011]. As in enterprise application integration (EAI), on top of well-defined and established processes, automation and integration of tools are most beneficial. This observation holds true for the test process as well. Our hypothesis is that applying the successful integration concepts from the EAI field [Peischl et al. 2011] and integrating various testing tools will result in increased efficiency in software testing [Turlo and Safronov 2011].

Workflow and Practical Tips for Integrating MBT in a Project

In this section we describe the steps required to successfully introduce MBT into your project and also provide some practical tips. But before reading on, you should be aware of the first pre-requisite for applying test automation in general and MBT in particular: a well-defined software development process that guarantees the adequate management of project tasks, requirements, test artifacts and change requests. If the process in place is not at the required maturity level, there is a strong probability that MBT will not work. Fast and reliable communication channels between the testers and the rest of the project team are an essential feature of such processes (see Figure 2):

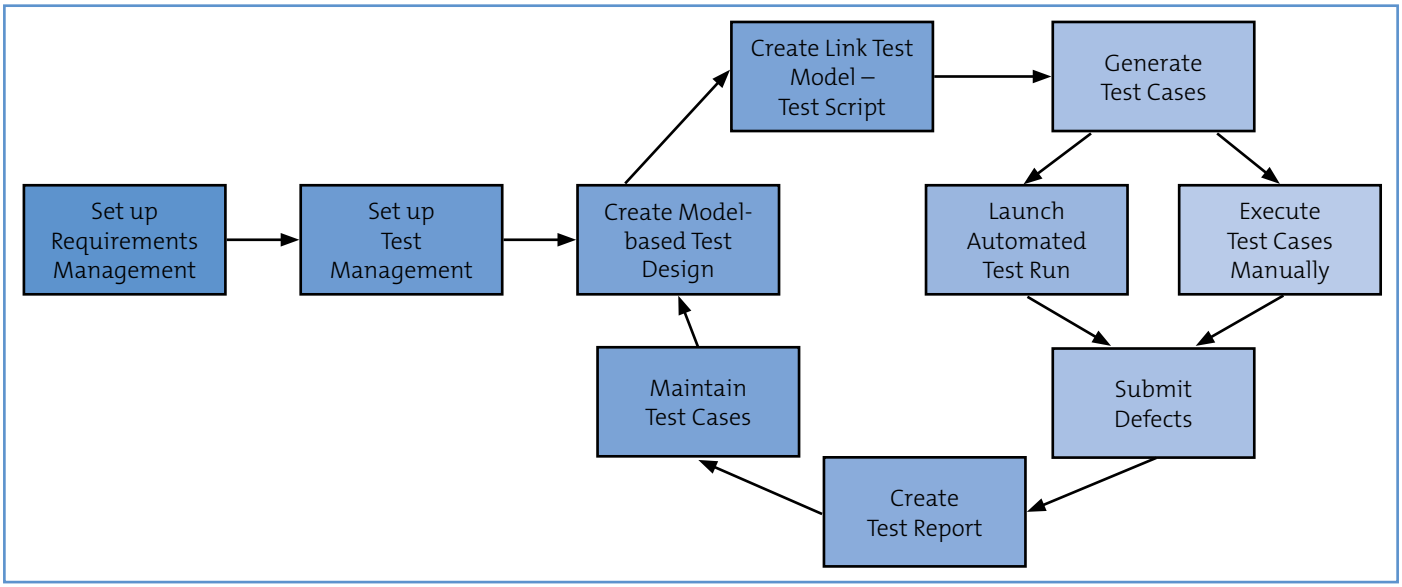


Figure 2: Workflow for integrating MBT into a project

1. Set up requirements management.

- A pre-requisite for MBT is that the requirements are detailed and clear enough that a formal model can be derived from them.
- Try to use activity diagrams, state diagrams, usage scenarios etc. instead of plain text.
- Make sure to plan interfaces that can later be used by test automation tools to access the system under test.
- Also involve test experts in early reviews in order to avoid testability problems later on.

2. Set up test management.

- Based on the requirements, create a test structure consisting of test packages and test cases. Don't only focus on the required functions, also plan tests for the non-functional requirements.
- Decide for which test cases model-based design shall be used. Typical candidates are tests for requirements that can easily be formalized as a model.
- For some requirements, MBT might not make sense. In addition, don't forget to plan some experience-based tests in addition to the systematic tests. A test model is a good thing but it cannot replace human experience and intuition!

3. Create model-based test design.

- A proper integration with the test management tool is important, e.g., for importing information about requirements and test packages
- Avoid reusing a model that has also been used for code generation. If the code and the tests are generated from the same model, no deviations will be found! Try to create a specific model from the testing perspective instead.
- Creating a test model is a challenging task that requires many different skills (test design, abstraction, understanding of requirements, technical background). Don't expect end users to perform this task but employ specialists instead!
- Start modelling at an abstract level, then add more detail step by step
- Make sure that requirements engineers and software designers are available to clarify any questions from the test designers.
- Pay attention to the reusability and maintainability of the model components, e.g., use parametrizable building blocks
- Try to cover both test data and test sequence related aspects in your model (e.g. define equivalence partitions for test data, activity diagrams for sequences). Finally, connect the two aspects by specifying which data is used in which step of the sequence.

4. Create links between the test model and executable test script

- If using a professional test generator that generates complete scripts (e.g., TEMPPO Designer), record GUI information and assign steps to GUI elements.
- If using a generator that only produces abstract keywords, implement an adapter or a script for each keyword.

5. Generate test cases

- Choose test coverage based on model type and criticality of the associated requirements
- Combine different test methods to increase defect detection potential. Use random generation in addition to systematic methods.
- The integration should make it possible to transfer the generated test cases directly to the test management tool.

6. Execute test cases

- In some projects, the test generator produces textual instructions for manual testing. However, more frequently, the test execution is performed automatically by a tool.
- The test execution tool should be fully integrated into the test management framework. It should be possible to start test runs and store the results automatically from within the test management tool.
- Verify if the generated test cases work as expected (GUI object recognition, timing, etc.). Correct the model where necessary until the tests are running smoothly.

7. Submit defects

- In cases of deviation from the expected behavior, it has to be determined whether the problem was really caused by a bug in the software or conversely by a defect in the test model.
- The flawless integration between test execution, test management, and defect management is necessary to efficiently submit and resolve new defects.

8. Create test report

- The execution logs created during the test run have to be transformed into a concise test report that informs the responsible managers about the current project status. For MBT it is important that statistics of the model coverage are included.
- Many test management tools can generate such reports, provided that they have access to all the required data. Again, a good integration with other tools is important.

9. Maintain test cases

- For each new version that has to be tested, the tests have to be adapted and repeated.
- It is essential that the test team is informed about all changes in the system under test. Even small modifications that are invisible to the user can cause an automated test run to fail.
- If designed properly, only some parts of the model have to be adapted. Afterwards, all test cases can be updated automatically by re-generating them.

Examples of Successful Tool Integration

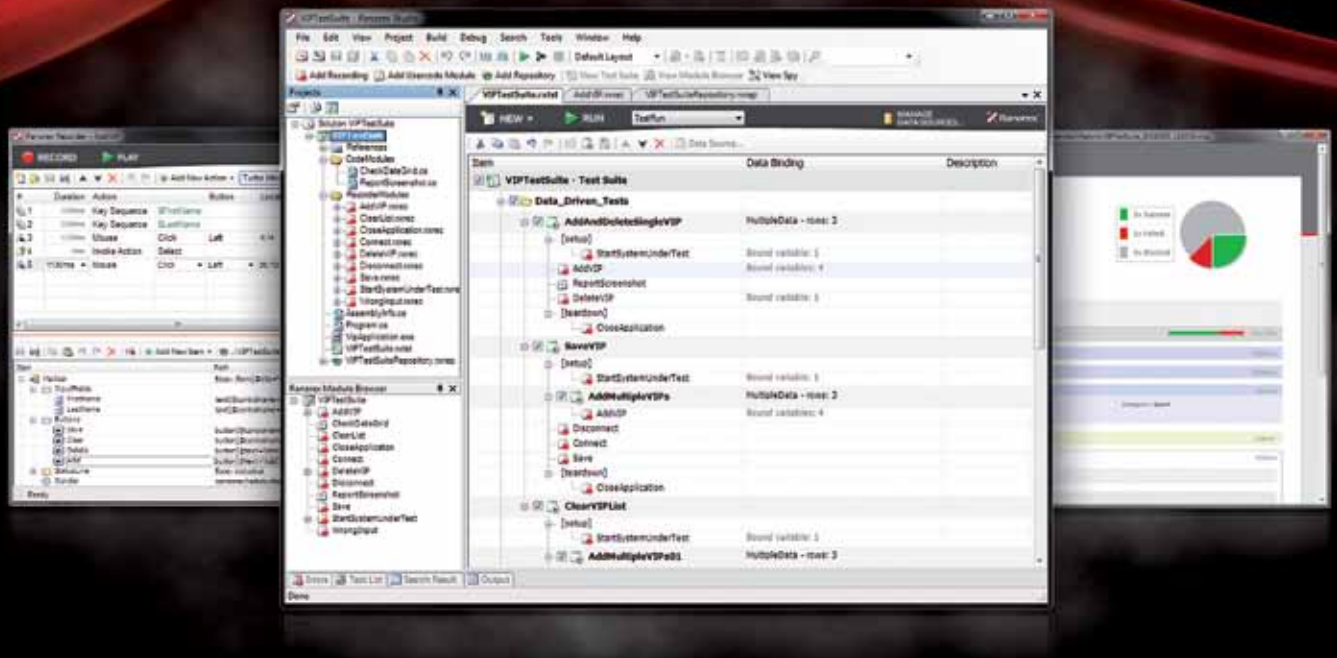
Automated Testing Control in Action – Octopus Automated Testing Control System

Let us consider a typical testing infrastructure of a software development company. Assuming that the build process is automated, it includes a build system, a file server, a versioning control system, a bug-tracking system, a test automation tool and virtual and/or physical machines. Integrating these elements of

Ranorex®

Automate your User Interface Testing

> Ranorex is the *Best Commercial Functional Automated Test Tool* for .NET and Flash/Flex applications.
>
> — *2011 ATI Automation Honors Awards*



- Use connectors for data-driven tests
- Build robust test automation frameworks
- Generate EXEs for pure flexibility
- Write custom code in C# or VB.NET

Record and Edit Reliable Test Actions

Manage and Execute Your Automated Tests

Reproduce Bugs and Maintain Your Tests

Award-winning test automation tools, which allow testing of many different application types, including: Web 2.0, WPF, Flash/Flex, Silverlight, Qt, .NET, 3rd Party Controls, and Java.

Download your 30-day Trial at **www.ranorex.com**

test infrastructure under a common user interface (UI) will produce a number of benefits, both in terms of testing process enhancement and cost cuttings. The purpose is not to create a universal infrastructure solution, but rather to link together existing software applications with the help of a control system that can automate interactions between them and provide a single UI for easy test configuration and management.

Today in the majority of companies applying automated testing, these systems and tools are autonomous, and the parameterization of test execution is done manually or using configuration files. When a tester receives a notification of build completion, he or she starts preparing the environment: configures virtual (or physical) machines, copies files, tests and programs from the file server to a test machine, and installs the application. Then, the tester runs a test suite, and reports detected bugs to the bug tracking system

The attempts of small and medium companies to realize a testing control system that fully automates this process are often incomplete or even fail completely as testing demands expertise and significant investments. Applied Systems Ltd., a software services provider for a leading European manufacturer of emission analysis and measurement systems, is one of the organizations demonstrating a success story.

The organization develops a complex customizable platform with hundreds of modules and agents and multiple configurations. With at least three produced builds per day, stringent quality requirements and limited testing resources, Applied Systems had an important challenge: to ensure regression testing is performed on every build. Six years of experience in automated testing and several attempts later, the company has finally implemented a closed-cycle system, named Octopus, will run assigned tests on every produced build without tester intervention.

The first release of Octopus had interfaces for the software tools used at Applied Systems: Microsoft® Team Foundation Server 2008 as a build machine, version control system, and defect management tool, Microsoft® Hyper-V as a virtualization server; and HP QuickTest® and Microsoft® Visual Studio as test automation tools. In the following version the most popular free software was added: ThoughtWorks CruiseControl.NET (build system), CollabNet Subversion (version control), VMware Server 2.0 (virtualization server), Mantis Bug Tracker and Bugzilla (bug tracking systems), and AutoIt (automation tool). Further plans include expanding this list and supplementing it with case generation tools.

If you are ready to undertake the challenge of realizing a similar solution in your company, take into account the following key principles for designing its architecture:

- Consolidate all tools engaged in testing via application programming interfaces (APIs) in a single UI. The front-end interface will serve as a convenient proxy for flexible parameterization of test execution. As a result, the testers will be spared the trouble of adjusting configuration files, transferring them from one directory to another, and switching between different applications.
- From the very beginning, provide for a scalable and expandable architecture of the system for automated testing control. On the one hand, testing volume is likely to increase over time and you will need to add more resources like virtual machines; on the other hand, the company might upgrade to a new version of the bug-tracking system or add a new test automation tool.
- To ensure the correct environment for test run execution, enable automatic system recovery of the test lab. In the case of virtualization, apply snapshoting technology to revert your virtual machines (VMs) to a specific state of the environment. To roll back a physical machine to a clean state, utilize the VHD (virtual hard disk) technology.
- It is useful if you realize the conversion of test results from "native" format into a well-structured, uniform cumulative report in HTML format. First, it is easier to quickly view where the bottlenecks are; second, a manager or customer will be able to access the report without having to install additional software; third, the results are immediately available at any moment during test execution.
- To ensure closed cycle testing, develop algorithms that perform tasks related to testing: queue the new builds for testing, launch VMs, prepare the environment, start test scripts, report defects to the bug tracking system, and generate a test report. Automatic submission of errors is a very useful feature in data-driven testing where each incorrect result should be registered. All the events should be logged to make the testing system transparent and to facilitate debugging.

Building your own system for automated testing control is not easy, but the payoff makes it worthwhile, especially if your company works on long-term projects. The organization will save time on non-productive tasks, make effective use of limited resources and enjoy improved testing processes. In the case of Applied Systems, automatic preparation of the environment resulted in saving two hours of tester time per day via automatic bug submission – at least a minute per reported defect. The execution of a test run on four concurrent VMs permitted the probing of each build with regression tests with the same amount of physical and human resources.

Integrating MBT into the Tool Chain – Atos TEMPPO Designer (IDATG)

While it is relatively easy to find tools for automated test execution, there are signigicantly fewer that focus on test generation. They can be categorized according to the method used for test generation: *data-oriented* tools and *sequence-oriented* tools. A tool that unites both features and can also be easily integrated into a project's tool framework is **Atos TEMPPO Designer**; this tool was previously called IDATG (Integrating Design and Automated Test case Generation).

When the research project IDATG started in 1997 the aim was to develop a simple tool for facilitating the maintenance of SilkTest® and WinRunner® scripts for GUI testing. Over the years, IDATG was steadily expanded to support an increasing number of test methods and output formats as well as testing via non-GUI interfaces. In 2004, IDATG became part of the test framework of the European Space Agency ESA (this success story can be found in Dorothy Graham's new book "*Experiences of Test Automation*" [Graham et al. 2012]).

Interfaces for Model Creation:

Many MBT tools do not have a model editor of their own, but instead rely on other sources from which models are imported, however, reusing models that have not been created with a focus

on testing has several disadvantages. Quite often such models are not formal enough (e.g. plain text annotations) or they contain a lot of information that is irrelevant to testing while missing important data. To overcome these problems, TEMPPO Designer (see Figure 3) has a built-in model editor that provides an independent way of creating test models. It uses a specific notation that is simple to learn and focuses on the necessities of testing. Test sequences are represented as *task flow diagrams* that consist of simple test steps (blue) and parameterizable building blocks (yellow) that represent reusable sub-sequences of steps. For GUI testing, steps can be assigned to GUI objects using the built-in GUI Spy.

Apart from common testing methods like equivalence partitioning, a number of innovative test strategies have been developed for the tool. These include CECIL (Cause-Effect Coverage Incorporating Linear boundaries) [Beer & Mohacsi 2008] and a hybrid algorithm for random testing [Mohacsi & Wallner 2010].

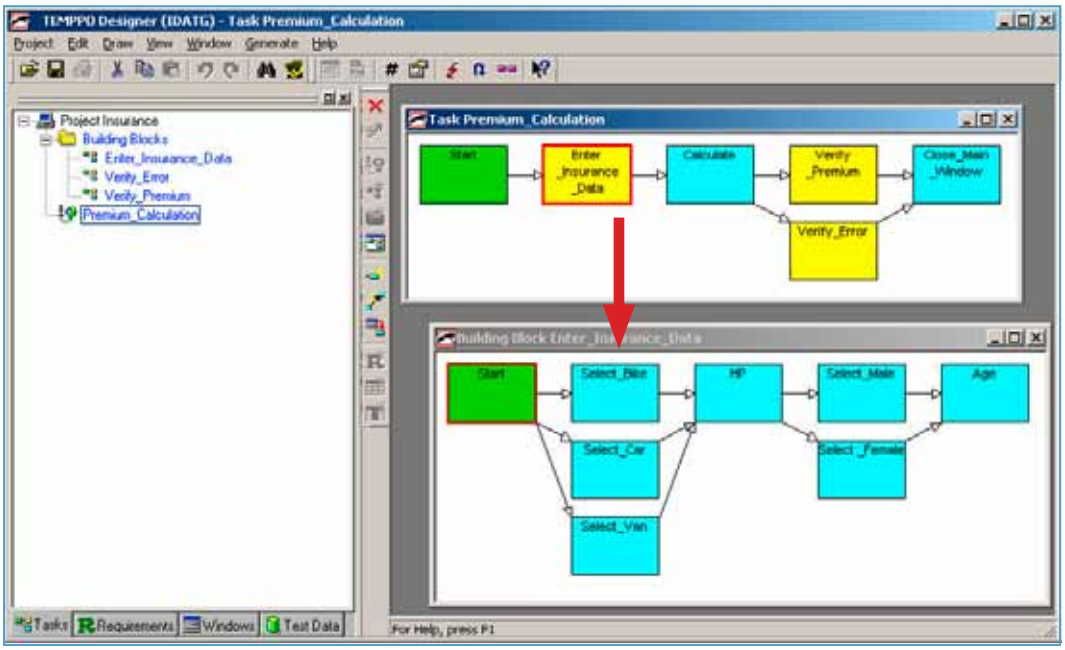


Figure 3: Task-Flow models in TEMPPO Designer.

Interfaces for Test Execution:

A major drawback of many test generators is that they only produce abstract test cases that still require manual completion, for instance by implementing a script for each keyword. A distinguishing feature of TEMPPO Designer is that its output are complete, executable scripts for a variety of popular test execution tools including HP QuickTest Professional® and the Micro Focus tools SilkTest® and TestPartner®. However, the tool can also be used for producing test cases that are executed manually or over via a non-GUI interface.

This direct integration between test generation and execution has proven to be a considerable advantage in that it delivers a significant reduction in test maintenance costs; i.e. instead of having to update each single test script for every new version of the system under test, it usually suffices to change a few building blocks inside TEMPPO Designer and let it generate new test cases. A case study from an ESA project has shown that the additional effort for introducing model-based test automation paid off after only four test repetitions [Graham et al. 2012].

Interfaces for Test Management:

Integration of test generation with test and requirements management is equally important. As its new name suggests, TEMPPO Designer is a part of the TEMPPO suite that also includes the test management tool **TEMPPO Test Manager**. Information can be passed in both directions: test structures and requirements created in Test Manager are passed to Designer. After the test model has been created, the generated test cases are passed back to TEMPPO Test Manager which ensures proper versioning, traceability, execution planning, and reporting.

There is an interface for HP Quality Center® that allows the user to import TEMPPO Test Designer test cases via Excel. Future plans focus on an even stronger integration of TEMPPO Designer with test execution tools, especially with Ranorex® and HP QuickTest®. For instance, it will be possible to import and reuse test sequences and GUI information recorded with Ranorex® in order to accelerate model creation and increase the number of supported GUI technologies even further.

Integrating GUI Test Automation – Ranorex Test Automation Framework

The Ranorex test automation framework is used to automate tests for web-based applications and client-based applications by simulating user actions through the graphic user interface (GUI). Ranorex can easily be integrated into continuous integration systems as Ranorex Studio creates an executable from a test suite project (see Figure 4).

This executable can be triggered to run by the system integrating the Ranorex test.

It is also possible to run one specific test case, execute a predefined run configuration and, set global parameters if the test project follows the data driven approach: there are many other options available when executing the Ranorex Test Suite using command line arguments.

To evaluate the outcome of the test, the integrating system usually examines the return value of the executable or its output text (e.g. "TEST FAILED" for failure). With Ranorex the return value "0" signals the successful execution of the test script and a return value of "-1" signals a failure in execution. Additionally, each test run can be accompanied by an XML-based report file providing detailed information for each single step executed within a test case.

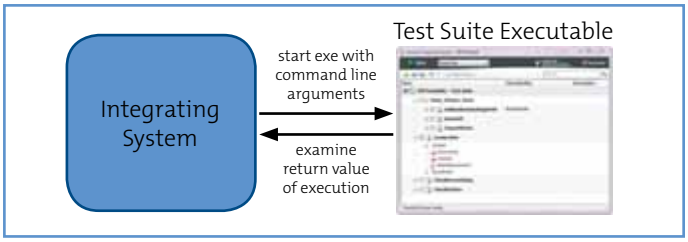


Figure 4: Integration of (generated) test cases with Ranorex Test Automation Suite.

Conclusion

In this article we discuss the impacts of an inadequate software testing infrastructure and show how model-based testing (MBT) can be efficiently integrated into the tool landscape of a test project. MBT has gained a solid foundation in recent years. Nonetheless, in order to fulfill its primary promise of reducing costs by detecting faults at an early stage and reducing test maintenance effort, it is necessary to professionally integrate MBT with other tools in your project (e.g. for test control and execution, test management, and requirements management). Our survey indicates that poorly integrated test infrastructure prevents professional testers from focusing their activities on more intellectual tasks such as the creation of models for generating test cases. We summarize the key factors that have to be taken into account for integration and point out successful integration scenarios relating to three concrete tools: the model-based test design and generation tool TEMPPO Designer (IDATG), the system for automated testing control known as Octopus, and the Ranorex GUI test automation tool. Professional tool integration will free test engineers from burdensome manual activities, thus paving the way for maximizing the benefits from MBT in your test project.

Acknowledgement

The work presented herein has been partially carried out within the competence network Softnet Austria II (www.soft-net.at, COMET K-Projekt) and funded by the Austrian Federal Ministry of Economy, Family and Youth (bmwfj), the province of Styria, the Steirische Wirtschaftsförderungsgesellschaft mbH. (SFG), and the city of Vienna in support of the Center for Innovation and Technology (ZIT).

References

[NIST 2002] The Economic Impacts of Inadequate Infrastructure for Software Testing, National Institute of Standards & Technology, Program Office Strategic Planning and Economic Analysis Group, May 2002.

[Turlo & Safronau 2011] V. Turlo and V. Safronau, Dealing with Challenges of Automating Test Execution, Proceedings of the Third IEEE International Conference on Advances in System Testing and Validation Lifecycle, October 2011.

[TE 2011] Improving the Test Process, Testing experience – The Magazine for Professional Testers, www.testingexperience.com, June 2011, ISSN 1866-5705.

[van Ewijk 2011] A. van Ewijk, TPI Next, Geschäftsbasierte Verbesserung des Testprozesses, dpunkt Verlag ISBN-10: 3898646858.

[Peischl et al. 2011] B. Peischl, R. Ramler, T. Ziebermayr, S. Mohacsi, C. Preschern, Requirements and Solutions for Tool Integration in Software Test Automation, Proceedings of the Third IEEE International Conference on Advances in System Testing and Validation Lifecycle, October 2011.

[Mohacsi & Wallner 2010] S. Mohacsi and J. Wallner, A Hybrid Approach to Model-Based Random Testing, Proceedings of the Second International Conference on Advances in System Testing and Validation Lifecycle, 2010, pp. 10-15, 22.-27. August 2010.

[Hüttermann 2011] M. Hütterman, Agile ALM – Lightweight tools and Agile strategies, August 2011, ISBN 9781935182634.

[Beer & Mohacsi 2008] A. Beer and S. Mohacsi, Efficient Test Data Generation for Variables with Complex Dependencies, Proceedings of the IEEE ICST, Lillehammer, Norway, 2008.

[Graham et al. 2012] D. Graham and M. Fewster, Experiences of Test Automation, Chapter 9, p. 155-175, Addison-Wesley, 2012.

> biography

Bernhard Peischl
is the coordinator of the competence network Softnet Austria. He is responsible for managing the network's R&D activities and is in charge of a number innovation projects dealing with software testing, verification and debugging. Bernhard received a MS in telecommunications engineering (2001) and a Ph.D in computer science (2004) from the Technische Universität Graz, Austria.

Rudolf Ramler
is key researcher and project manager at the Software Competence Center Hagenberg, Austria. His research interests include software testing, quality management and measurement. Rudolf works as a consultant in industry projects and is a lecturer with the Upper Austria University of Applied Sciences and the Vienna University of Technology. He studied Business Informatics and holds a M.Sc. (2001) from the Johannes Kepler University of Linz, Austria.

Vitalina Turlo
is a Product Manager at Applied Systems Ltd. She holds an M.A. in International Business from Belarus State Economic University and has the international background in business analysis and research through her experience of studying and working in the United States and South Korea. Vitalina is in charge of coordinating the work of Applied Systems' automated testing competence center and crafting a product strategy for Octopus.

Stefan Mohacsi
joined Siemens in 1997 and became project manager of the research project IDATG (Integrating Design and Automated Test Case Generation). Its most notable application has been as a core part of the test automation framework of the European Space Agency (ESA). Today, Stefan is senior consultant for model-based testing at Atos. In addition, he is a member of the Austrian Testing Board and has held numerous lectures at international test conferences.

Valery Safronau
is the Head of Testing department at Applied Systems Ltd. He specializes in building and streamlining testing processes, improving test infrastructure, developing automated tests using different frameworks. He is the author and architect of Octopus system for automated testing control. Valery was a speaker at numerous specialized software testing events.

Tobias Walter
is studying Informatics at Technical University of Graz. Besides studying he is working at Ranorex as technical support engineer. Next to his support activities he is also writing articles published in Ranorex blog and Ranorex user guide.

Enabling the information society

BCS
The Chartered Institute for IT
First Floor, Block D
North Star House
North Star Avenue
Swindon SN2 1FA

T +44 (0) 1793 417 417
F +44 (0) 1793 417 444
E pressoffice@hq.bcs.org.uk
www.bcs.org

08 February 2012

BCS works with iSQI to offer Certified Agile Tester

BCS, The Chartered Institute for IT, has signed an agreement with the International Software Quality Institute (ISQI GmbH) to be an exam provider for the Certified Agile Tester certificate.

Michiel van der Voort, Director of International and Professional Development, BCS, The Chartered Institute for IT, says: "We're delighted to be working with iSQI. Software testers who work on agile technology projects are in demand as businesses strive to maintain their competitive edge, so this certificate is ideal for anyone working or wanting to work in this important field."

The foundation level certificate was launched by iSQI in Spring 2011 and is aimed at software testers working in an agile project environment or those interested in agile, such as managers, Scrum masters and developers.

Stephan Goericke, Managing Director iSQI says: "It is great to be working with BCS, The Chartered Institute for IT to deliver our certification. Agile projects are an increasing element of many businesses and this certification provides an ideal practical platform of knowledge and skills for software testers working in this area."

The certificate is awarded to successful candidates who complete a 5 day training course which features practical exercises, continuous soft skills assessment throughout the training, followed by theoretical and practical exams.

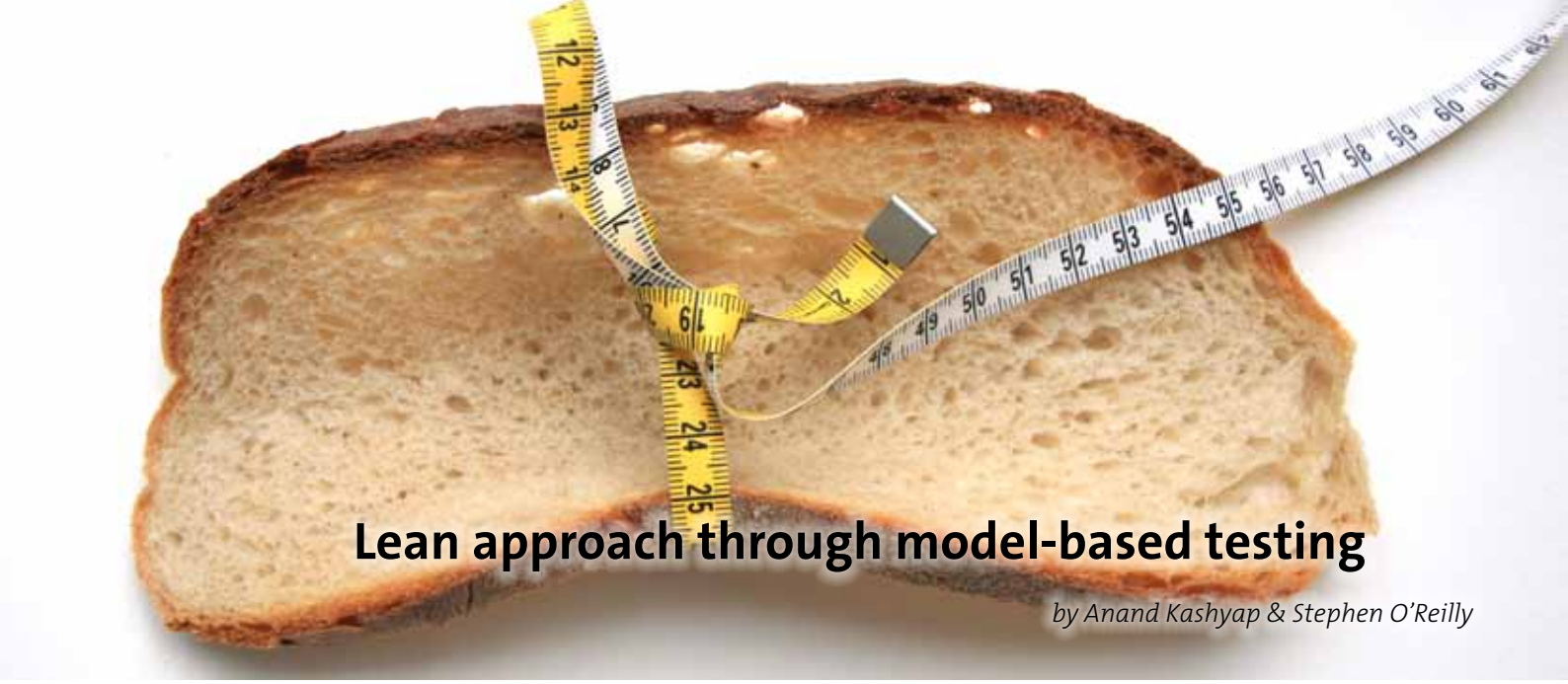
Michiel adds: "This certificate has been developed in conjunction with subject matter experts and, as the Chartered Institute for IT, we will be aligning it with SFIA (the Skills Framework for the Information Age)."

The certificate will feature as part of a series of forthcoming roadshows visiting London (13th February), Leeds (14th February) and Birmingham (15th February) aimed at giving IT professionals the opportunity to learn more about the subject.

Further information can be found at: www.agile-tester.org

Editors

For further information
Lynda Feeley MCIPR
Head of Press & PR
BCS The Chartered Institute for IT
First Floor, Block D, North Star House, North Star Avenue, Swindon SN2 1FA
Tel: +44 (0) 1793 417 417 | Direct Dial: +44 (0) 1793 417 711 |
lynda.feeley@hq.bcs.org.uk | www.bcs.org

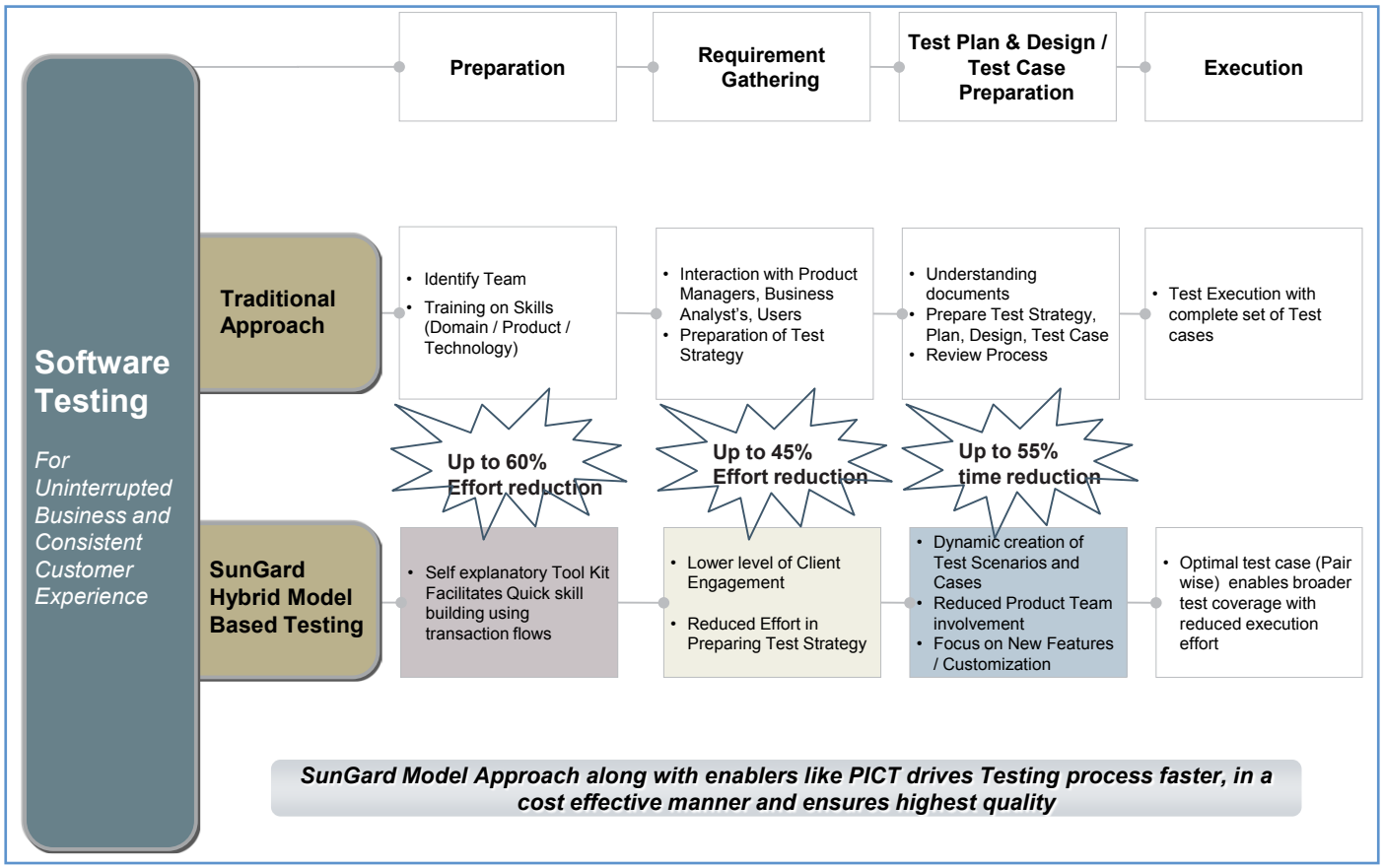


Lean approach through model-based testing

by Anand Kashyap & Stephen O'Reilly

Much of the focus on implementing Lean (a philosophy aimed at reducing waste and introducing continuous improvement and agility to meet customers' needs) today is pointed towards application development. Many vendors aim to sell different kinds of application toolkits targeted at specific technology stacks. These application toolkits have allowed developers to rapidly create advanced software programs directed at multiple domains. Developing these programs, however, is only part of the process in getting a product ready for market. Once the product development cycle is underway, developers have to deal with multiple issues such as deployment, integration and testing. Unfortunately, the typical amount of effort spent on design & development is not always applied to other areas of the Software Development Life Cycle (SDLC) such as testing and, depending on the nature, complexity and risk to the user of the software being designed; this can be deemed to be acceptable at times.

Historically, testing has been the SDLC's poor relation and is often considered an unwanted overhead where the least amount of bandwidth or budget is provided for this type of activity. Testing though, is a highly specialized activity that can be saddled with many challenges such as:



SunGard Test Accelerator (SunGard's hybrid model-based testing approach)

- Insufficient time, budget, resources
- Lack of suitable tools, standards, process
- Ambiguous requirements
- Lack of understanding around the importance and long-term benefits of testing

To overcome such challenges, SunGard developed a Test Accelerator using the model-based testing (MBT) methodology that aligns with a Lean philosophy. MBT uses physical models of the system being tested to represent the required behavior of the system based on business requirements.

SunGard's Test Accelerator provides a tool kit for the speedy generation of MBT test cases that are optimized utilizing pair-wise data generation and auto-generation of business keywords – ultimately aiding "on the fly" automated test execution. The toolkit facilitates an improved knowledge transfer and retention mechanism by representing the functional flow within a graphical interface.

The toolkit also provides an interface for an online review of the artifacts created and enables the test cases and business keywords to be easily re-generated when a review is conducted and, where comments are required, to be incorporated into the models created. This reduces the overall turnaround time as part of a more rigid review process.

By merging two or more models, the application is tested incrementally and thus ensures sufficient regression coverage. Meanwhile, by optimizing and prioritizing the test cases generated, testing becomes more flexible with definitive coverage against the schedule.

SunGard Test Accelerator (SunGard's hybrid model-based testing approach)

Objective

The objective of the MBT testing approach and toolkit developed at SunGard is its alignment with Lean methodologies and the benefits that such an approach can provide. It has been developed based on real-life experiences and customer demands and is designed to be implemented in parallel with product development from the initial capture of test requirements through to test execution, while also providing the flexibility needed thereafter to respond to product changes.

Definition of lean approach and benefits of model-based testing

This approach has been described by the Lean Enterprise Institute as a way of

"... Maximizing customer value while minimizing waste" (2008)

while Green Suppliers Network described the approach as being

"....a systematic approach to identifying and eliminating waste (non-value added activities) through continuous improvement by flowing the product at the pull of the customer in pursuit of perfection" (2009).

In the world of software development, Lean can be loosely interpreted as "harmonizing business and technology" by nurturing optimization and being sensitive to customers' needs and competitiveness while, at the same time, being flexible enough to respond to changes. Typically, the approach consists of the following elements:

Standardization

Standardization can be described as being one of the most powerful lean tools. By documenting best practices, standardization forms the baseline for continuous improvement. As the standard is improved, the new standard becomes the baseline for further improvements, and so on. Improving on standardization is a never-ending process.

Efficiency

The definition of efficiency is being "able to produce something with the minimum amount of time and resources". Efficiency in Lean, however, has a few caveats.

- Firstly, efficiency in Lean is aimed at making sure that a common understanding exists within the team
- Secondly, it is to ensure that there is no confusion between the terms "efficiency" and "effectiveness"

Being efficient is generally good, but being **efficient** by not doing the right things, is not being effective!

For example, perhaps test case design can be completed quickly and efficiently. So good, that a great supply of test cases are created waiting to be executed. It is essential in Lean that such efficiency gains are targeted at areas where the work is desired and effort is not incorrectly/inefficiently expended focusing on wrong areas.

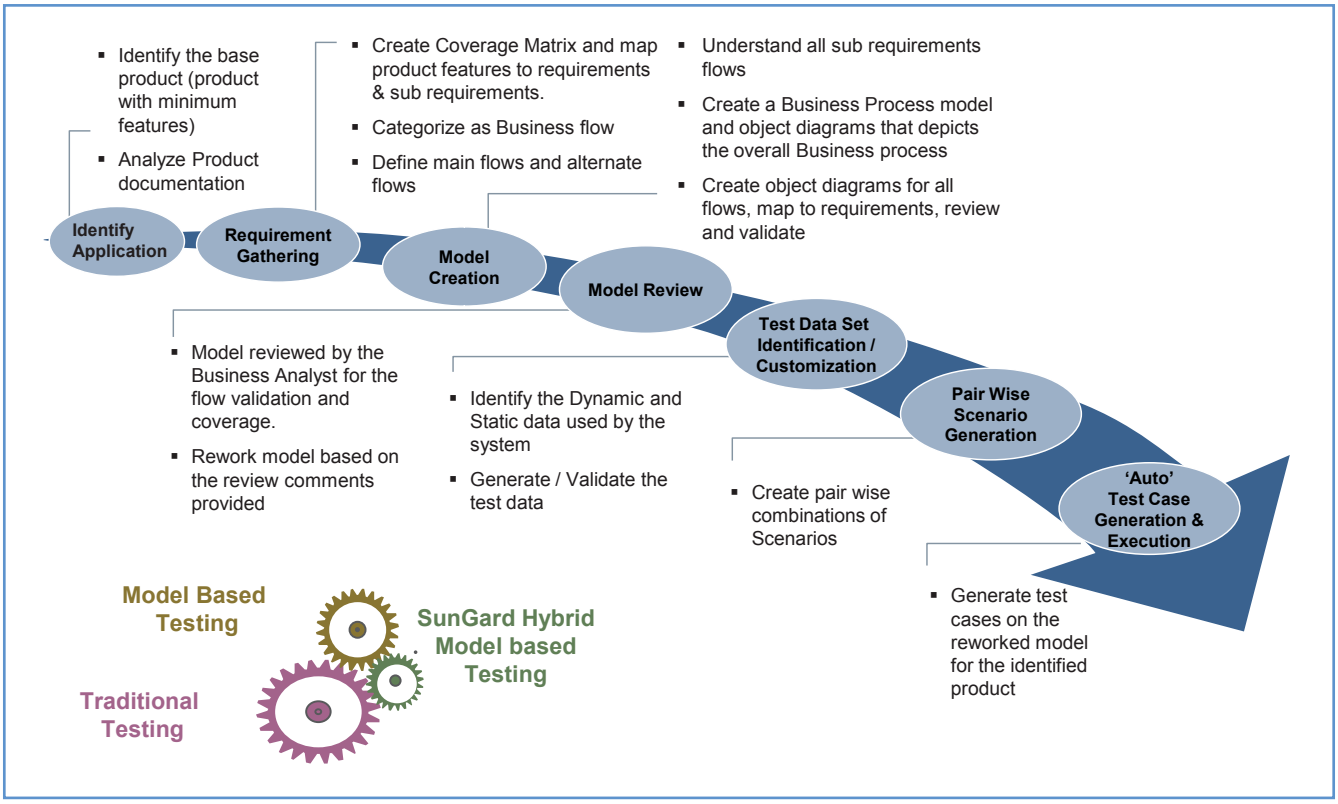
Cost reduction

The cost benefit in Lean philosophy is achieved by eliminating waste and being efficient and effective. So, optimizing the entire SDLC provides the desired cost benefits compared to the implementation of Lean or MBT approaches for purely an isolated activity.

Test management approach

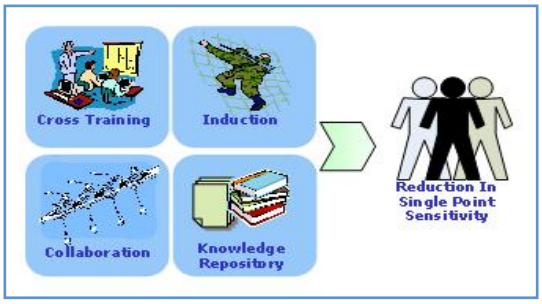
With today's challenges around testing, and with budgets for QA being lowered regularly, there is potential for a reduction in the quality of deliverables in order to complete the assigned tasks with more demanding timelines and fewer resources. The solution? Be creative by aiming to achieve the same results using less money or resources! Optimizing one phase of any activity will not provide the desired result. Adopting Lean philosophy and creating solutions using a Lean/MBT approach, provides the desired effect but needs to be applied throughout the testing life cycle.

In order to achieve best results, SunGard applies the Lean philosophy to traditional test management with minor modifications to the traditional approach. The following diagram explains the test management approach adopted for Lean philosophy using MBT.



Knowledge management using MBT

SunGard utilizes a knowledge retention approach complemented by an effective knowledge management process. The figure below depicts the knowledge retention strategy that is followed.



How quality of deliverables should be managed as part of a Lean/MBT approach

Documenting best practices is definitely the starting point. This aids in ensuring a consistent quality of deliverables. However, standardization, as mentioned, will not ensure improved quality on its own. All artifacts created need to go through a well-defined review process and the level of rework effort should be kept minimal. A root cause analysis / causal analysis and resolution process should be rigidly adhered to and updated periodically.

Testing using models leads to:

- Enhanced specifications
- Improved, more pertinent metrics
- Integration of testers into the development process at an earlier stage
- Continuous testing
- Quality deliverables
- Predictability and risk reduction
- Efficiency and Effectiveness

How this approach aligns with agile processes

SunGard's Lean/MBT approach complements the agile process by providing an early test design phase. The testing team utilizes process flows (activity diagrams); decision tables and/or object diagrams created by the development team to generate logical test cases. Once the development for an identified iteration is completed, the test cases generated using the MBT approach are executed by the test team; at the same time the test design team is generating test cases for the next iteration by utilizing the process flows created by the development team.

Reviews for the models created are conducted in conjunction with the development team and stakeholders, thus ensuring uniformity in what is being developed and tested.

Models created for particular iterations are merged incrementally in order to create the regression pack, and results for iterations are updated incrementally.

A business keyword driven framework developed by SunGard assists thereafter in automated test execution using the business keywords produced through a Test Accelerator.

Short Lean/MBT testing case study

Challenge

The primary objective of this project was to model and generate tests for a Graphical User Interface (GUI) window in a large market watch application. The tests needed to verify that the application responded appropriately to inputs.

The User Interface (UI) had six menus, each of which comprised several menu items. In addition, the main menu had different sub-menu items.

Approach

The first attempt employed a single relation with 44 constraints. However, the constraints were difficult to understand and were eventually found to have too many invalid combinations. Subsequently, a two-way model with 9 constraints was created. One major issue encountered centered on the fact that all menu items essentially consisted of different values of the same variable, whilst only one menu item could be selected for a particular scenario.

Ultimately, the understanding of the requirements and determination of the constraints proved to be the most difficult part in creating the model. The requirements were undocumented, so therefore we had to utilize the application to understand the finer points of the implementation. This, however, was expected since typically many well-established products in the market do not have well defined requirements or are quite often incomplete. This is one of the main reasons for creating self-explanatory models which can also be re-used for knowledge transition and knowledge retention.

Results

Test generation from the model yielded 25 scenarios with 408 pair-wise interactions (test cases). The output of the model was then used to execute tests in a logical sequence and a business keyword framework was developed for automated test case execution.

Output analysis was simple because it consisted solely of confirming that the correct menu item and corresponding window were displayed after an action key was clicked. A test for the correct menu item and corresponding window was implemented in the business keyword framework, and the automation engine was checked after each test to determine the actual result.

> biography

Anand Kashyap
manages the testing program for SunGard Global Technology's Testing Service Line in India. With over 12 years' experience in the industry, he has successfully performed various roles including solution crafting, requirements gathering, requirements analysis, test planning & estimation, test execution and management, project tracking and scheduling, tool evaluations, test automation, technical approaches, release management and project management activities for multiple technologies across various domains.

Stephen O'Reilly
is Head of the Quality Assurance (QA) & Testing Practice for SunGard Global Services in Ireland. With over 16 years' experience in the industry, he is a key member of the Global Services Testing Service Line steering group and is responsible for establishing and promoting Acceptance Test Accelerator (ATA) – a SunGard testing service offering aimed at eliminating pain points around requirements gathering and product/solutions testing for internal product development teams and external customers.

First model, then test!

by Ben Visser & Bart Vrenegoor

Testing traditionally consists of much manual, repeated work. Automation of testing is known to fail, especially when employed without a good approach. Perhaps failures are even greater in number than successes? Model based testing has often been looked upon as 'one step beyond', modeling followed by automated test execution. This is even less likely to succeed, as these models often contain the same inconsistencies as the text based designs used to create them.

We know that many of the defects we find during testing can be traced back to requirements and functional documentation. So why should we even bother using these design artifacts to create models for test automation? Is that really useful? Testers need to leave the notion of traditional model-based testing behind and embrace a test paradigm in which models are used for other test activities. In this paradigm, models are used even before test preparation starts; a paradigm where we first model and then test. We call this 'model-based services'.

The first step of this paradigm shift is to apply model-based reviewing and after that model-based test design. Testers will use models to improve their test preparation and test specification (meaning: faster, cheaper, better!), while laying a foundation for automated test execution at the same time. Projects that implemented full model-based test services saved up to 70% test cost and 90% turnaround time in the test specification phase!

Model based reviewing

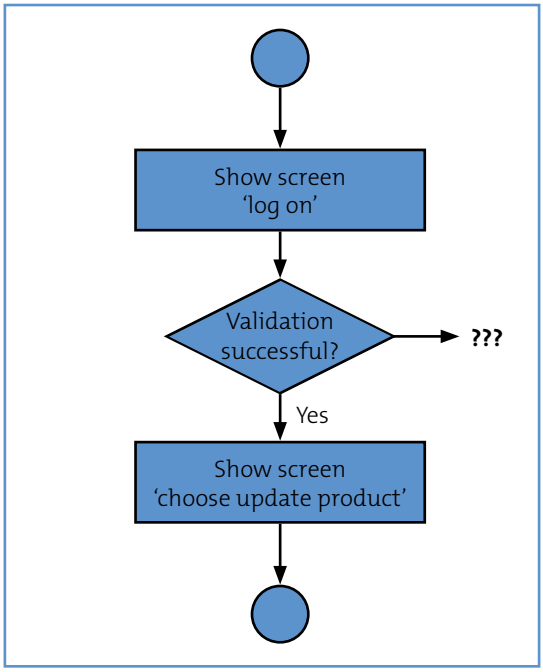
According to TMap NEXT®, one of the first test engineering activities in any test project is 'intake test base'; the test engineer reads the design artifacts to get familiar with the system he or she will be testing, and/or to verify the quality of the artifacts, i.e. can they be used as starting point for test design. This activity can be very informal and aimed at 'only' understanding the subject matter, without trying to – at this stage – improving anything, or it can be very formal with elaborate inspections (e.g. Fagan inspections) specifically undertaken to detect and remove as many flaws as possible in the design already at this stage. Model based reviewing aims at both: actively seeking knowledge and understanding of the subject matter, and revealing as many as flaws as possible. The type of flaws the tester focuses on are inconsistencies in structure, the type of knowledge and understanding the tester focuses on is suitability for testing.

Model based reviewing follows a three step approach:

- Testers create a model from the (text based) design artifacts.
- The models are discussed and completed in collaboration with designers and analysts.
- The models are finalized and are suited to be used as an unambiguous base for test design.

Creating the model

The first step is to create a model from the (text based) artifacts. A good, easy to learn way of modeling is flow modeling. Flow testing (also called path coverage) is very mature; it's a test design technique to derive test cases from a flow and it's extensively described in test literature. Tools to verify the quality of a flow model are readily available on the Internet.



Let's assume that a tester performs a review using a flow model. While reviewing, he might run into the following text. It's a very simple example from a functional design, be it a use case or some other design document:

> ***"After opening the web site the 'log on' screen is shown in which the user can enter his username/password. After validation the user is shown the 'choose update product' screen".***

The point is not that this description says nothing about how to validate the username/password combination. This may very well – and for good reason – be written down in some other artifact. What's missing here is what should happen if the username/password combination does not compute. Then what? Back to the log on screen? Most probably, but how many times? Infinitely, or is there some security rule that enforces a maximum of three tries after which a suspend mechanism must be invoked?

One could say that a tester, especially if he's well-trained and experienced, should and will find this flaw anyway. That's true. However, if a tester, under time pressure, has to read through a design artifact of dozens of pages, he's bound to miss some of the flaws. Testers are not perfect. And even when he picks it up, it might be very tempting to think (to know!?) that after three tries, the suspend mechanism must be invoked. So he does not question the artifact. But that's interpreting the design. Testers are human!

Discussing the model

A typical product of a traditional review is a list of comments and questions. In practice, it's a list of design flaws and errors. Let's be honest, this can be perceived rather negative by the designer. With model-based reviewing, a shift is made towards the positive.

If a tester can't put the design into a proper flow, there may be a couple of reasons. Firstly, the tester may not have the proper background to do so – he just lacks subject matter knowledge – or it has to do with the design itself – too little detail, inconsistent, incomplete, ... – or a combination of factors. In any case, something needs to be done, measures need to be taken.

The best way forward is to discuss the design with all parties involved. With a traditional review, designers may take remarks personally, an attack on 'their work'. Model based reviewing will take the sting out of this 'design discussion': you're not questioning the work of someone else, you're putting your own model up for questioning! Showing that you created a model out of a design artifact itself will prove you've invested time and effort into understanding what another designer created.

Suppose a tester wants to discuss the example mentioned before. The tester can now use the process flow as a communication tool with the designers of the functional description. The goal is to complete the model by asking questions like: 'What happens if validation fails? ', or 'What will happen if validation fails over and over again?' In practice, both designers and testers will be adjusting the model in no time. After the model is finished, the tester needs to emphasize that all changes to the model imply changes to the original artifact. It's up to designers to make these changes.

Finalizing the model

As stated earlier, model-based reviewing takes place at the stage 'intake test base'. An important aspect of this stage is a common understanding of the scope of the testing at hand. Merely agreeing on testing functional design 'FU_update_product v0.9.3' or use case 'UC001.C version 1.4' isn't enough. To what extent will this be tested? The finalized model serves as a reference for the agreements on this. If all paths of the flow model will be tested, only the test depth needs to be decided, but if the design is too elaborate to test all paths, including all alternate paths, and every possible error handling path, then a different way of drawing a path (different coloring, dotted or not) can be applied to indicate whether or not that (part of the) path is included in the test at hand.

Model based test design

Model based reviewing is time consuming, and up to this point no test case is created and no test is executed. Therefore model-based test *design steps* in. Models are unambiguous (at least good models are). So a test suite derived from such an unambiguous model consists of a predictable set of test cases, provided by a formal test design technique. One of the nice features of the Internet is that it hosts an increasing number of automated test design tools. With the help of these tools, a test suite can be built out of models in a matter of minutes, where manual test case specification would easily take days or more. Thus, the extra effort put into creating models during test preparation is won back several times over during the test specification phase!

Advantages of model-based test services

Applying models in our day to day test activities shows several benefits:

For test preparation:

- Creating and reviewing models not only finds defects in the test base, but also enhances our understanding of the application under test
- It will make the test base more clear and objective.
- Using models removes the 'personal element' from reviewing.

For test specification:

- Accessibility and traceability back to the test base of the test suite increase.
- The width and depth (scope) of the test suite becomes much clearer, since it is the scope of the model.
- Test specification might very well be automated.

It's plausible that model-based services also provide advantages for test execution. Increasing the quality of documentation by applying model-based reviewing has two advantages. First it's likely that the quality of the delivered software increases, as the improved documentation is used by programmers as well. This could speed up test execution. Furthermore, a major barrier is removed for test automation as the inconsistencies in documentation are now gone.

And this last point was exactly the reason why the usefulness of model-based testing and test automation in general were questioned earlier in this article. With model-based services, however, this is no longer relevant! By using models early during the review process, quality is added as early as possible, the test specification phase is accelerated and doors are opened for successful test automation. Therefore: First model, then test!

> biography

Ben Visser
is a highly experienced test manager and test consultant. In a career of over 15 years as a tester, he has fulfilled a wide range of test functions, from programmer of automated scripts, to test manager of large international programs. He has worked in traditional waterfall developments as a change manager responsible for test and acceptance environments, as well as model-based testing. Based on this extensive experience, he co-authored Sogeti's TMap NEXT® Business Driven Test Management (Nov. 2008) and more recently TPI® NEXT (Nov. 2009).

Bart Vrenegoor
is a Research & Development Consultant. He's an expert on several test & QA related topics, like Model-Based Testing, testing of Business Intelligence and TMap NEXT®. He's an international speaker, coach, and author on these subjects. Besides his work as a R&D consultant, Bart works as a test manager.

Belgium Testing Days

March 12–14, 2012
Brussels, Belgium

Belgium's most popular international 3-day testing event

www.belgiumtestingdays.com

Watch out for Frenemies

by Ian Moyse

Ever received an email unsolicited (yes, we know you get spam) to your work or personal email address? Found an account of yours compromised that you can't log in? Gone onto messenger and had messages from someone you don't know popping up? So just how do they do that? How do they know who you are, and how do they get your email or messenger IDs, and is there anything you can truly do about this?

I often see an email in my Hotmail inbox from someone I know that looks out of context, not something I would expect them to send. Perhaps a "have you seen this picture of yourself on site HTTP://zzyahs.com/images", or "I saw this great product and you must take a look at it – check it out at www.greatproductexample.com". It is very likely that what has happened here is that either their mail account itself has been compromised, or their email address has been harvested and a spoof email (an email that looks like it's from them but isn't) has been created.

So your friends start receiving emails , messenger messages or even Facebook messages from what purports and looks to be you, but is not! Who is really communicating with them, and how did they get your identity so easily? These are malicious attackers riding on the back of your personal equity with that person, to entice them into trusting the message they have just received, from what they think is you. Why are the attackers taking this approach? A knock at your front door and you look out the window, a stranger you may ask for ID, not answer the door or act on alert; someone you recognize you're going to open the door and welcome them in, right!? This is exactly the reason why in the electronic world they are taking this approach of disguising themselves as you and knocking on your friends' electronic doors.

The difference is that at your real door it is very difficult, nigh impossible, for someone to disguise themselves to really pass as a friend you have known for years – electronically this becomes easy. Getting a person's details often is surprisingly easy, check-out for example people you know on Linkedin and Facebook alone and see how quickly you can gather their personal details. You will be surprised how many post their mobile number onto their linked in profile and often their full DOB and marital status. On Linkedin, married women often even put their maiden name in brackets in their ID name (so that old friends can still find them), but of course the maiden name is often a security identifier used for authentication. Now check their Facebook profile, often linked to from their Linkedin profile, assisting you in your profiling, and quickly you can find yourself with a ton of personal data on someone and also details of their friends, too.

It's not far from this to take a further step and steal their identity, launch a malware attack onto them through a malicious or fake Linkedin email request (an email looking like a Linkedin connection but from a fake link and site). Once you have one login of your victim, you're away and can often piggy back from this into obtaining further logins they may have with similar IDs or the same password. Now send a message to all their friends who will more than likely believe the message, link or payload as being from a known trusted friend and hence drop their guard, click away and hence themselves become infected or compromised.

Ever been on a messenger client such as MSN and found random people adding you? Often as a guy it will be a sexy sounding female with a picture of a girl looking like a model – too good to be true – YES, of course it is!!

This is a growth area called SPIM (spam over instant messenger) where your email ID has been harvested from perhaps your facebook profile, Linkedin posting or even a posting on a discussion board where you have left your email address openly visible. You then get a random 'linkee' request to chat and – hey presto – they have you hooked into a dialog. During this, you will often get a link presented saying something like "hey, did you see the picture of you I saw / posted here http://example.co.uk", and of course clicking here to see it leads you to a drive by infection."

Add to this we are now seeing text message spam and even recently instances of Blackberry messenger spam and fake contact additions, where someone tries to add you on Blackberry messenger for example, similar to the MSN example above. In all these instances, the attackers are preying on your trust in real people as your vulnerability. Once they compromise one of your IDs, it often leads to a snowball effect, as they either use that to attack your friends from your account or utilize it to get into other accounts

Belgium Testing Days

Belgium's most popular international 3-day testing event

This year's theme

QA versus Testing! Antagonism or Symbiosis?

43 speakers delivering "High-Level Knowledge Transfer"

5 tutorials





27 tracks
4 keynotes
1 exciting lightning talk

4 workshops

Exhibitors & Supporters

coralius · bcs The Chartered Institute for IT · ps_testware Your devil's advocate · quasus we make it valuable · MICRO FOCUS · SOGETI · iSQI · hp · Key Performance Application Performance Management Solutions · PARASOFT We make software work. · jamo solutions · the test leaders · te testing experience · Agile Record · Díaz Hilterscheic · AQIS

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
www.diazhilterscheid.de

AQIS bvba

Uilstraat 76
3300 Sint-Margriete-Houtem (Belgium)
Phone: +32 16 777420
Fax: +32 16 771851
www.aqis.eu

twitter
Follow us @BelgiumTD

of yours (for example, maybe you have the password reset for your Facebook stored in your gmail folders – once into gmail they have this, too!) All too many users also still use the same password on multiple sites for their own simplicity, of course this is useful until one of your logins is exposed and then they all become at risk.

If they have access to your main email account, also consider this: that they can then perform a password reset on other sites sending it to your mail account, changing the password and in effect not only copying your login, but replacing it so they and not you can get into your accounts, locking you out of your own access!

Do not assume that an electronically received message from a known email address, Facebook ID or messenger name is the person you think it is! Do not trust it any more than you would someone unknown when receiving odd links and suggestions to go to web sites. Do not assume that a message is safe just because it appears to be coming from a friend. Ensure you have the best malware protection enabled on your device and that you remain cautious. Far too many are falling foul to these social masquerading scams and lose personal data, privacy and money! The attackers know this and will not stop, but will continue to prey on your trust!

> biography

Ian Moyse
has 25 years experience in the IT sector. He is a published speaker on Cloud computing and sits on the board of Eurocloud UK, the governance board of the Cloud Industry Forum (CIF). Ian has been keynote speaker at many events and has introduced cloud security to a range of leading companies. Ian was awarded global 'AllBusiness Sales AllStar Award for 2010' and the 'European Channel Personality of the Year Award for 2011'.

EUROPE 2012

Testing & Finance

May 16–17, 2012
in London, UK

The Conference for Testing & Finance Professionals

www.testingfinance.com

Time	Track 1	Track 2	Vendor Track
08:00	Registration		
09:10	Opening Speech José Díaz		
09:15	**Keynote:** **"Testing and Finance in Agile Projects, Social Media and Regulation"** Krishna Rajan, Barclays		
10:15	Break		
10:20	**"Model based testing using acceptance tests as oracle"** Adrian Rapan	**"Testing the Financial cloud"** Bart Knaack	
11:10	Coffee Break		
11:25	**"What's so special about Testing Financial applications, anyway?"** Bernie Berger	**"Help!, my supplier works agile and I don't"** Chris C. Schotanus	
12:15	Break		
12:20	**Keynote:** **"The Future of Testing in Finance"** Paul Gerrard		
13:20	Lunch		
14:35	**"Visualising Quality"** David Evans & Gojko Adzic	**"Practical approaches to systematic test case design"** Dr. Hartwig Schwier	
15:25	Coffee Break		
15:40	**"Test specialist on agile teams: A New Paradigm for Testers"** Henrik Andersson	**"Experience in Multinational Bank with Advanced Stakeholder Value Requirements: Quantified for Testability and Tracking"** Tom & Kai Gilb	
16:30	Coffee Break		
16:45	**"Financial Software Testing"** Jean-Paul Varwijk	**"Identity fraud and the use of electronic identity"** Robin John Lee	
17:35	Break		
17:40	**Keynote:** **"Online Crooks, Governments, Banks, You and me!"** Peter Kleissner		
20:00	Cocktailparty		

We would like to invite you to the first **Testing & Finance** conference in **London, UK**.

The two-day conference **Testing & Finance**, which is held once a year and for the first time in the UK, brings together quality assurance specialists and experts from the financial world from both home and abroad. Since 2005, Díaz & Hilterscheid has been inviting well-known international speakers to this event.

"**Testing and Finance in Agile Projects, Social Media and Regulation**" is the main theme of next year's **Testing & Finance**, taking place on 16–17 May 2012 in London, UK.

The focus lies on technical subjects in Software Testing, as well as on Regulatory Reporting and its impact both from a business and technical perspective. Well-known organizations from the industry offer information about their products and services during the exhibition which is held parallel to the conference.

We hope you will enjoy the **Testing & Finance** conference 2012!

Time	Track 1	Track 2	Vendor Track
08:00	Registration		
09:15	**Keynote:** **"Agile Testing in a Regulated Environment"** Peter Varhol		
10:15	Break		
10:20	**"Test Data Management - Adressing Data Sensitivness and compliance without sacrificing productivity"** Dr. Klaus Haller	**"Specification by example' applied to Cpp legacy code using FitNesse Cslim"** Laurent Decorps	**"Assuring the success of Agile in a regulatory environment"** Matt Robson (CAPITA)[1]
11:10	Coffee Break		
11:25	**"Agile Transitioning - all change!?"** Matthew Steer	**"Filling the Gap - Testing what BDD doesn´t"** Matthew Archer	
12:15	Lunch		
13:30	**"Automated Testing For Financial Feeds Made Easy"** Glyn Rhodes & Stuart Walker	**"Co-operative Banking Group and Infosys' Approach for Successful QA Transformations"** Paul Shatwell & Kiruba Vijayaraka	
14:20	Coffee Break		
14:35	**"Ford´s Lean"** Jorrit-Jaap de Jong	**"Testing Credit Risk Rating Models"** Ron Coppejans	
15:25	Coffee Break		
15:40	**"ATDD with robotframework done right"** Sajjad Malang & Catherine Decrocq	**"Regression Testing Live Systems - Doing It Better"** Scott Summers	
16:30	Break		
16:35	**Keynote:** **"Back to the future: the changing role of QA"** Gojko Adzic		
17:35	Closing Session, José Díaz		

[1] 25-minute presentation

Please note that the program is subject to change.

Exhibitors & Supporters

If you want to support the Testing & Finance 2012 conference please contact us at info@testingfinance.com.

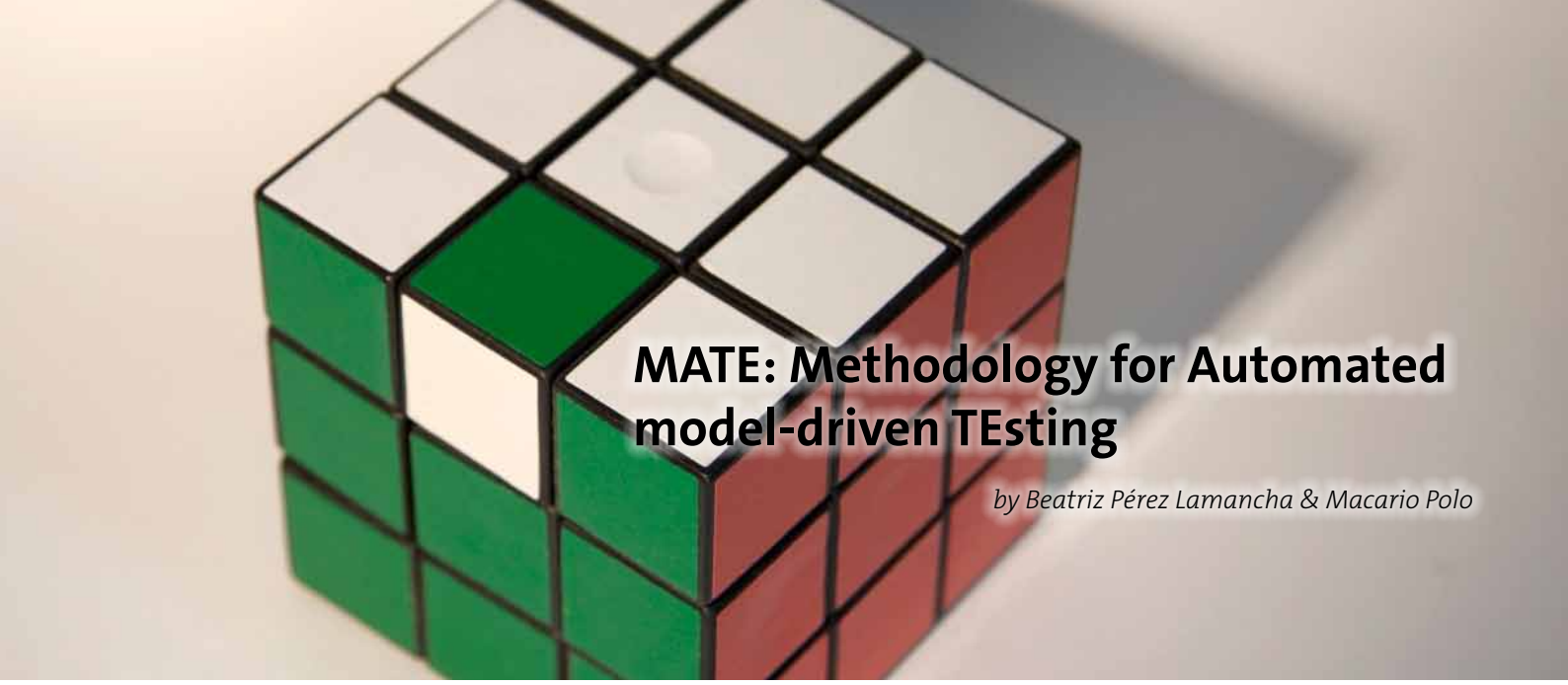
Testing & Finance Europe 2012 –
A Díaz & Hilterscheid Conference

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

info@testingfinance.com
www.testingfinance.com

www.xing.com/net/testingfinance



MATE: Methodology for Automated model-driven TEsting

by Beatriz Pérez Lamancha & Macario Polo

Model-Based Testing (MBT) provides techniques for the automatic generation of test cases using models extracted from software artifacts [1]. There are several approaches for model-based testing [2, 3]. Nonetheless, the adoption of model-based testing by industry remains low and signs of the anticipated research breakthrough are weak [4]. Model-Driven Testing (MDT) refers to model-based testing that follows the Model Driven Engineering paradigm, i.e., test cases at different abstraction levels are automatically generated using models extracted from software artifacts through model transformations.

This article presents the MATE methodology, which was defined to achieve the following goals:

- **Functional testing level**: The methodology generates test cases at functional level, i.e., the system is considered as a black box and the stimuli between the system and the exterior are tested.
- **UML notation**: MATE uses the UML meta-model [5] to represent requirements, design and testing models. Testing models are also represented using the UML Testing Profile (UTP) [6].
- **Model-driven test case generation**: Test cases are automatically generated from design models and evolve with the product up to the test code generation. System behavior is represented at design time using UML sequence diagrams, which are first transformed into test models and later into test code.
- **Standardized artifacts handling**: The framework is based on Object Management Group (OMG) standards. The standards used are UML [5], UML Testing Profile [6] as meta-models, and Query/View/Transformation (QVT) [7] and MOF2Text [8] as standardized transformation languages. A UML-compliant profile (Oracle Test Profile, OTP) is used to describe the oracle information.

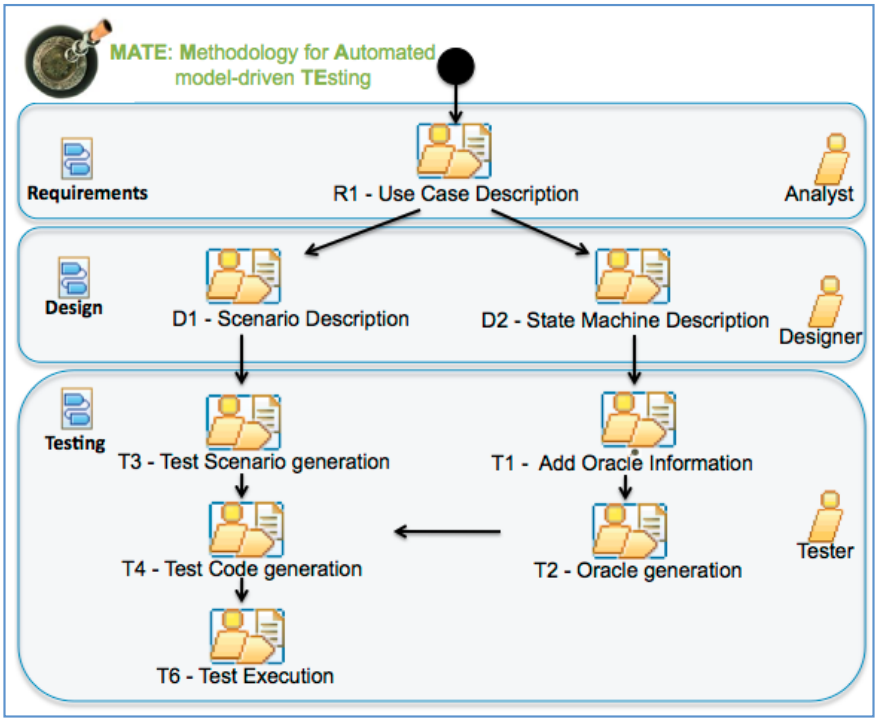


Figure 1

- **Implementation using existing tools**: Since the framework is completely based on standards, it can be adopted with almost any UML-compliant tool. Therefore no tools were developed to support the methodology.

MATE Overview

Figure 1 describes the activities defined in MATE and their relationships. Regardless of the software development process followed to develop the system, this work assumes that UML models are used in the requirements, design and testing disciplines. Figure 2 shows the artifacts and their dependencies, and Figure 3 shows an overview of the main work products for a simple functionality: Login. Arrows describe the transformations used to obtain one work product from another.

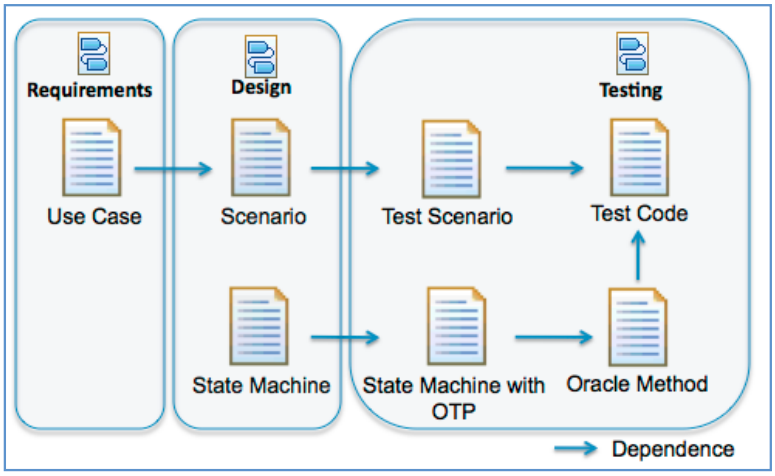


Figure 2

These work products are:

- **Use Case**: Use cases are used to capture the requirements, i.e., what a system is supposed to do.
- **Scenario**: Since use cases are specified using textual notation in natural language, we resorted to scenarios as a systematic approach to capture use cases. A scenario describes one path in the flow of a use case and can be described using UML sequence diagrams.
- **State machine**: A UML state machine represents a finite state machine using UML notation.
- **State machine with OTP**: The state machine is augmented using the Oracle Test Profile (OTP) with the testing information. In many cases, this allows the automatic addition of the oracle to test cases.
- **Test Scenario**: This represents the test requirements for a scenario. Test scenarios are described using the UML Testing Profile, representing the test case behavior with UML Sequence Diagrams.
- **Test Code**: Test scenarios are translated into executable test code that tests the code of the corresponding scenario.
- **Oracle Method**: Is automatically obtained from the State Machine with OTP. The test code uses the oracle method to calculate the expected result automatically.

Figure 3 shows the Login design scenario and the generated Login_test test scenario. The test scenario proceeds from the transformation Scenario2Test. Figure 3 also shows the Login_SM state machine, which represents the possible states for the Login

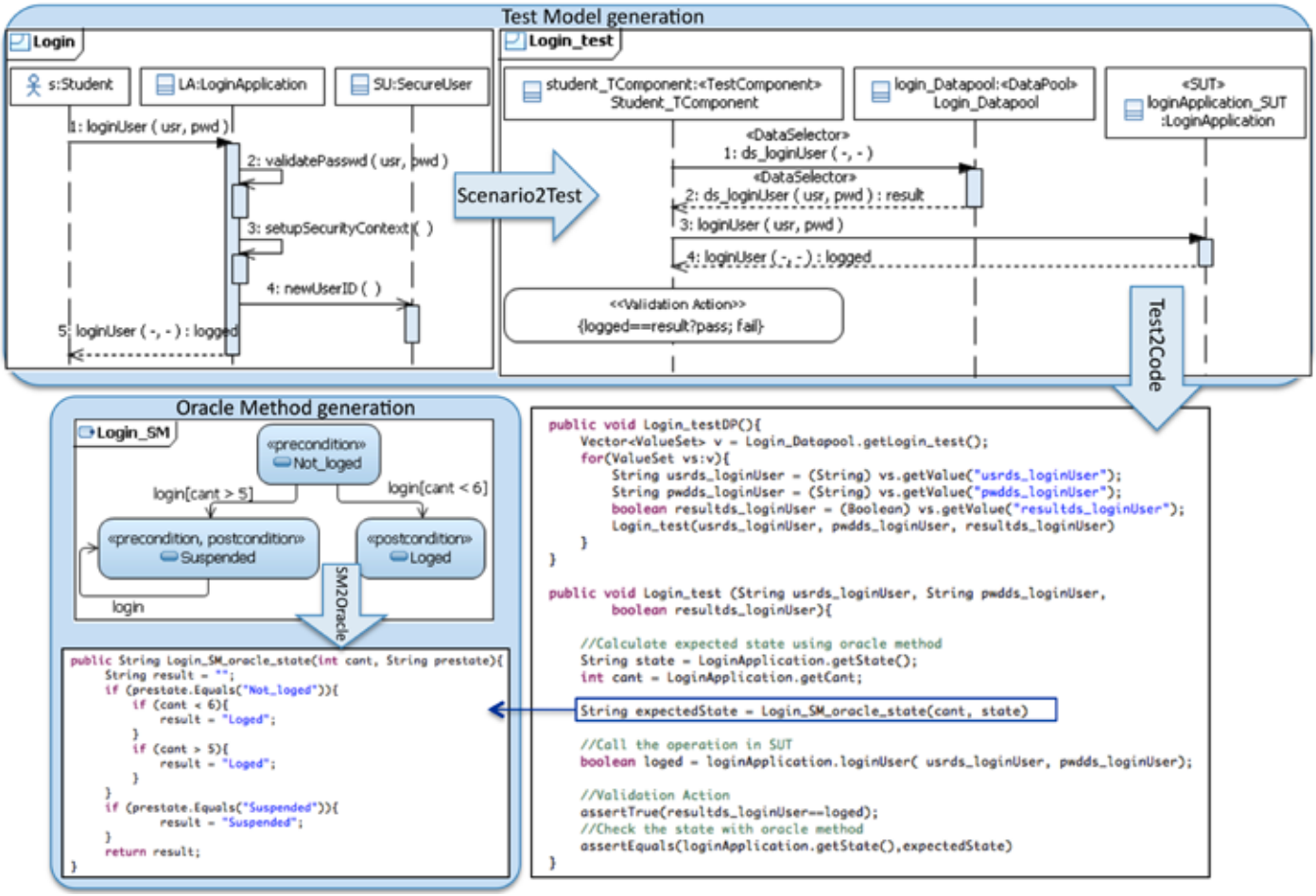


Figure 3

Prof van Testing recommends

Díaz Hilterscheid

twitter

Follow me @vanTesting

IREB – Certified Professional for Requirements Engineering – Foundation Level

Description

The training is aimed at personnel mainly involved in the tasks of software requirements engineering. The tutorial is designed to transfer the knowledge needed to pass the examination to become an IREB CPRE-FL.

After earning a CPRE-FL certificate, a certificate holder can assess whether a given situation calls for requirements engineering. He understands the fundamental characteristics of the discipline and the interplay of methodological approaches, e.g. interview techniques, description tools or forms of documentation.

More information regarding the required knowledge can be found in the IREB Syllabus, which can be downloaded from the IREB web site: http://www.certified-re.com

IREB International Requirements Engineering Board

Dates*	3 days
27.–29.03.12	Berlin/Germany (de)
06.–08.06.12	Berlin/Germany (de)
16.–18.04.12	Helsinki/Finland (en)
19.–21.06.12	Oslo/Norwegen (en)

*subject to modifications

Website:
http://training.diazhilterscheid.com

functionality. The Login_SM_oracle_state is the oracle method generated from Login_SM through the SM2Oracle transformation.

Figure 3 shows the JUnit code automatically generated through the Test2Code transformation. It includes two methods: Login_testDP (to get the test data) and Login_test (the test case). This method uses the oracle method to obtain the expected result.

MATE framework

The only requirement to apply the MATE methodology is the availability of tools that allow model edition, transformation and visualization. Several commercial and free UML modeling tools have improved editors to generate all the required models. The selected modeling tool must allow describing UML models and applying UML profiles, as well as to export and import XMI files following the UML meta-model, from which these models are built. Examples in this article are depicted using IBM Rational Software Architect.

Furthermore, core of the MATE methodology is model transformation, which MATE supports by means of the framework described in Figure 4. Transformations are implemented at these levels:

- **Scenario to test scenario**: The input for this model-to-model transformation is a scenario described with UML sequence diagrams. It produces a test model composed of a class diagram representing the test architecture (UTP test architecture) and a sequence diagram representing the test behavior (U2TP test behavior). The involved objects correspond to instances of the class diagram. This transformation is implemented using a QVT transformation and is represented as Scenario2Test in Figure 4. The test model obtained conforms to the UML Testing Profile. More information about this transformation can be found in [9].
- **Test scenario to test code**: This model-to-text transformation takes the test scenario as input and produces as output some JUnit executable test code to test the system. This transformation is implemented using a MOF2text transformation and is represented as Test2Code in Figure 4. More information about this transformation can be found in [10].
- **State machine to oracle method**: UML state machines are used to describe the test oracle; first, the state machine is augmented with specific information for testing using the Oracle Test Profile. Then, the model-to-text transformation translates the UML state machine (with the profile applied) into a method in the same language in which the application is developed; finally this method calculates the expected result. This transformation is implemented using MOF2text transformation and is represented as SM2Oracle in Figure 4.

Conclusions

MATE helps the tester in test case automation by using UML models that describe the system under test. The test model and its corresponding test code are derived following a model-driven approach. The result is a complete suite of executable test cases with the following information:

- **Parameterized input test data**: The tester stores the input test data in the dataPool, and the test case automatically retrieves it. The operation in the dataPool returns all the input data for that test case, and the test case is executed for each set of inputs.
- **Automated calculus of the expected result**: Two methods are obtained from an UML state machine: one to calculate the expected result, and another to get the expected result.
- **Test case procedure**: The test case procedure is automatically generated from the sequence diagram representing the functionality under test, and it uses the information about the test input and the expected result. The test case procedure focuses on functional testing. Then, the interaction between the system and the actors is tested. As a result, the JUnit generated method recovers the test input from the dataPool; this method calls the operations to test in the SUT with the test input, uses the oracle methods to calculate the expected result, and finally validates the presence or absence of errors by means of one or more assert XUnit statements.

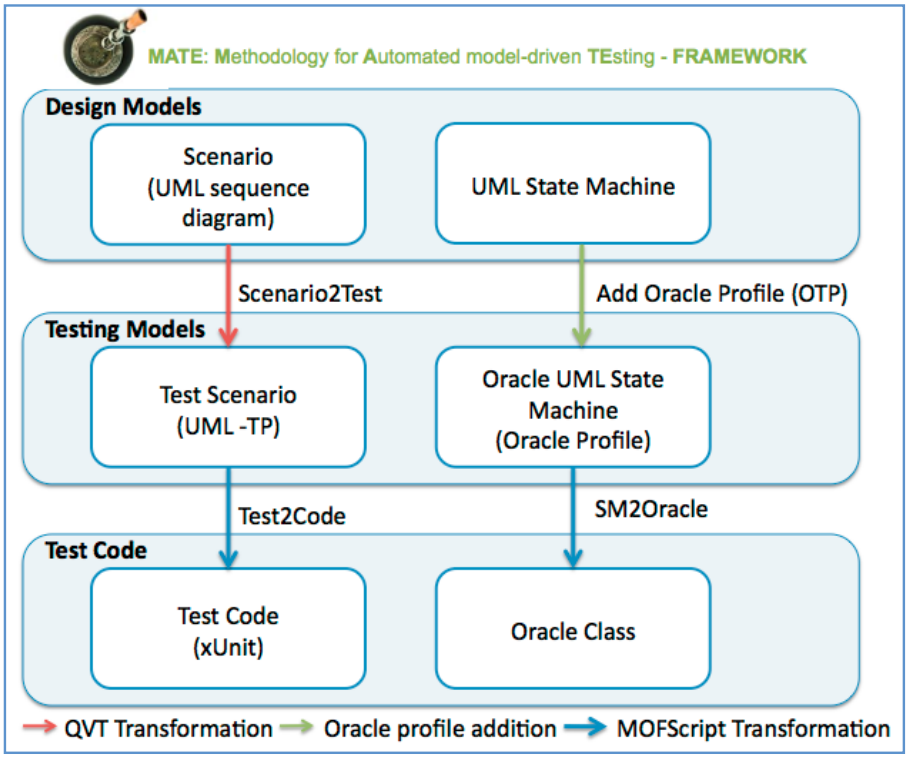


Figure 4

References

1. Dalal, S.R., et al. Model-based testing in practice. in International Conference on Software Engineering. 1999: IEEE.

2. Dias Neto, A.C., et al. A survey on model-based testing approaches: a systematic review. in Workshop on Empirical Assessment of Software Engineering Languages and Technologies. 2007: ACM.

3. Prasanna, M., et al., A survey on automatic test case generation. Academic Open Internet Journal, 2005. 15: p. 1-5.

4. Bertolino, A. Software testing research: Achievements, challenges, dreams. in Workshop on the Future of Software Engineering. 2007: IEEE.

5. OMG, Unified Modeling Language (UML), superstructure. 2007.

6. OMG, UML testing profile Version O.M. Group, Editor. 2005.

7. OMG, Meta Object Facility 2.0 Query/View/Transformation Specification. 2007.

8. OMG, MOF Model to Text Transformation Language. 2008, OMG.

9. Pérez Lamancha, B., et al. Automated Model-based Testing using the UML Testing Profile and QVT. in MODEVVA. 2009. USA.

10. Pérez Lamancha, B., et al. Model-Driven Testing: Transformations from test models to test code. in International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2011). 2011: SciTePres.

> biography

Beatriz Pérez Lamancha *received her Master in Computer Science at the PEDECIBA Program (Program for the Development of Basic Sciences), School of Engineering, Universidad de la República University, and her Engineering degree in Computer Science at the Universidad de la República (Uruguay). Currently, she is a Ph.D. student in the program of Advanced Informatic Technology of the Universidad de Castilla-La Mancha.*
She is Adjunct Professor in Software Engineering in the Universidad de la República, Uruguay. She was founder, consultant and project leader in the Software Testing Center (Centro de Ensayos de Software, CES) in Uruguay. The Software Testing Center was created as a public-private entrepreneurship between the Uruguayan Information Technology Chamber (CUTI) and the Engineering School from the Uruguayan Republic University. Its main contribution is the improvement in competitiveness and productivity in the Information Technology (IT) industry.

Macario Polo *is a professor of computer science at the Department of Information Systems and Technologies in the University of Castilla-La Mancha. He is also an active member of the Alarcos Research Group. His research interests relate to the automation of software testing tasks. Polo has a PhD in computer science from the University of Castilla-La Mancha. Contact him at macario.polo@uclm.es.*

Agile TESTING DAYS

November 19–22, 2012 in Potsdam, Germany
www.agiletestingdays.com

Become Agile Testing Days Sponsor 2012

We are currently setting up the program. We have recieved 198 papers for 42 scheduled talks!

We will also set up 10 tutorials, 9 keynotes, 8 sponsor talks, 2 demo dessions, 2 early keynotes, consensus talks, a test lab, open space, a test clinic, testing dojos and coding dojos.

For this phenomenal event, we are looking for supporters. Please have a look at our portfolio and create your own big sponsorship package:

Exhibitor

Diamond Exhibitor:	18.000,00 €
Platinum Exhibitor:	12.000,00 €
Gold Exhibitor:	6.000,00 €
Silver Exhibitor:	4.000,00 €

Conference Sponsor

MIATPP Award Trophy Sponsor:	3.300,00 €
Conference Bag Insert:	550,00 €
Coffee Break Cup Sponsor:	1.700,00 €
Social Event Sponsor:	10.000,00 €

Media Sponsor

Program First Page Advert:	1.100,00 €
Program Full Page Advert:	550,00 €
Program Half page Advert:	280,00 €

Session Sponsor

90 Minutes Product Demo:	2.500,00 €
60 Minutes Early Keynote:	2.500,00 €

Please find details on packages online at www.agiletestingdays.com

Exhibitors & Supporters 2011:

T-Systems · Seapine Software · NOKIA · XING · SEQIS The Austrian Software Test Experts! · telerik deliver more than expected

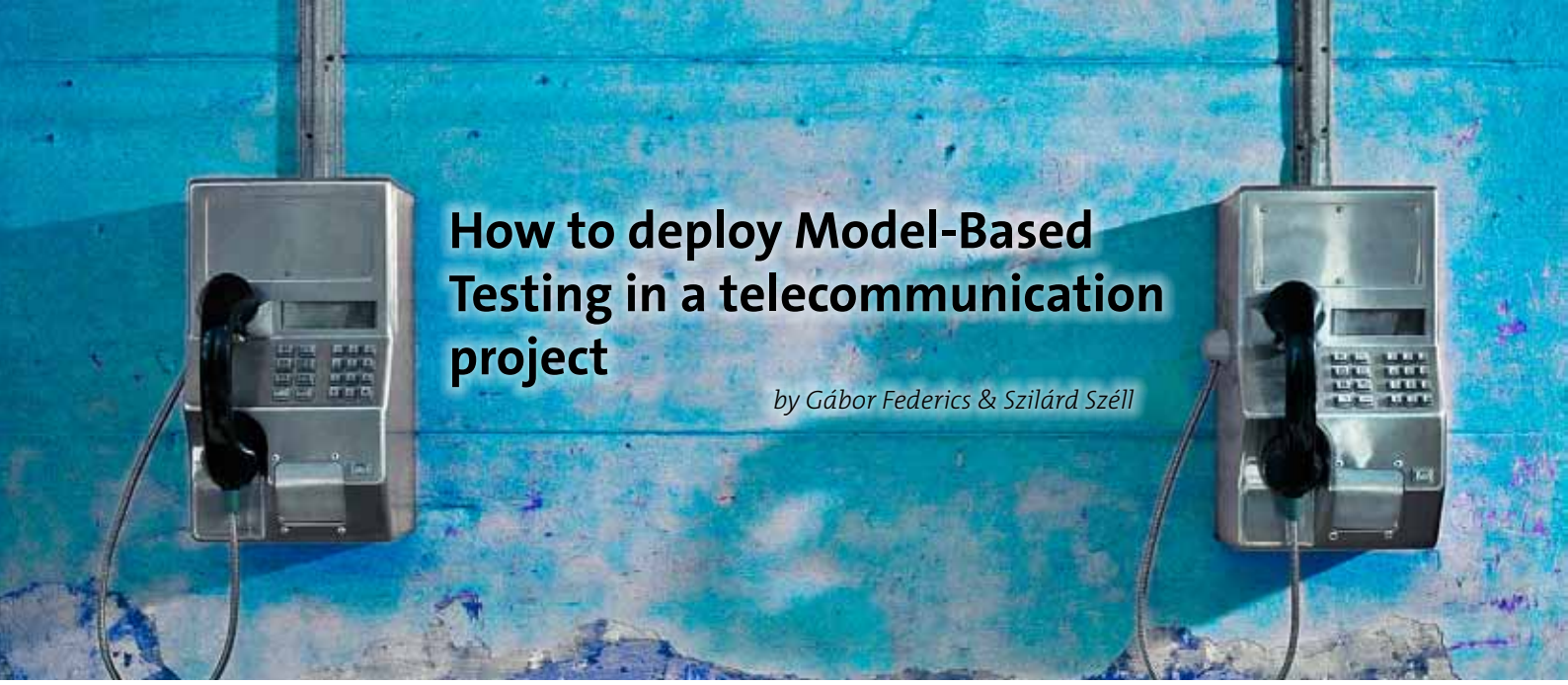
dpunkt.verlag · hp · iSQI · POTSDAM · JetBRAINS Develop with pleasure! · te testing experience · Agile Record · Díaz Hilterscheid

A Díaz & Hilterscheid Conference

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

info@agiletestingdays.com
www.agiletestingdays.com
www.xing.com/net/agiletesting
www.twitter.com/AgileTD



How to deploy Model-Based Testing in a telecommunication project

by Gábor Federics & Szilárd Széll

The challenge

As test manager you're expected to outperform; deliver more, faster, and better quality.

In today's telecommunication industry you must deliver early without sacrificing the required very high standards of quality. You should also follow the GSM and 3GPP standards, and keep in mind not to break any of the existing functionality developed in the last 20 years. This was the challenge we faced in 2009; we had to figure out how to make testing even more efficient both in terms of effort and on the fault finding capability, to reduce the time needed for testing of new and legacy functionality, and to increase the fault finding profile. So, to put it simply, we needed to speed up testing.

The solution

As test automation was already on a very good level, with test execution, analysis and reporting automated, we had to take the next step: automate what was still manual, the test design. On top of creating test scenarios automatically, we also wanted to provide the optimal test case set for a given coverage requirement, to automate documentation creation, and to provide executable test scripts compatible with the existing test harness.

Since 2008, Nokia Siemens Networks has been deeply involved in the European Community project called D-MINT, to develop the methodologies, tools and industrial experience to enable European industry to test more effectively and more efficiently. Seeing the internal demonstration of the achievements, we realized that Model-Based Testing (MBT) could help automate test design. We decided to check if MBT was feasible for us and if it was in line with our expectation to increase efficiency.

Pilot

First, we had to convince our engineers and management that MBT would be worth trying. We invited the MBT solution vendor we had used during D-MINT to introduce the technology, the ways it could help us, and to give a short overview of a standard deployment project. Having received a good overview of the technology, we decided on a pilot agreement. In 2010, we started to introduce the concept of Automated Test Design to provide hands-on experience using the vendor's tool, and to calculate return on investment figures.

Setting the goals of the pilot and the criteria for continuation was very important. We agreed on two goals: demonstrate that an automatically generated test suite can be integrated with the existing automated functional test environment in a reasonable time, and by comparison to previously available manually generated test suites assess what gains Automated Test Design can provide in areas of productivity, flexibility and maintainability. We also wanted to produce reusable assets – models and model components – for use in later deployments.

It took only three weeks for the vendor's solution engineers and our test engineers to achieve the first goal.

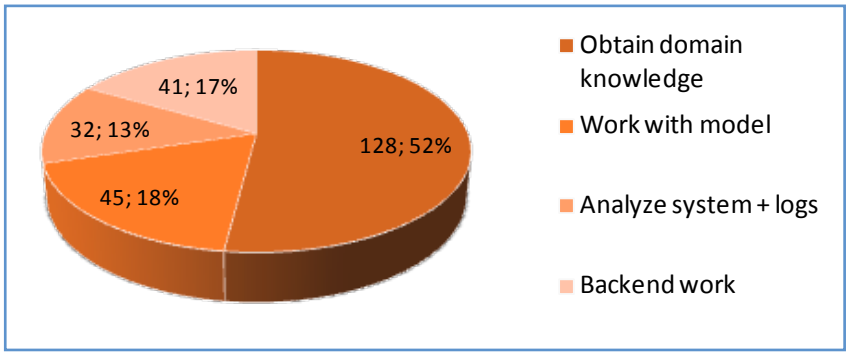


Figure 1. Effort share between different tasks during proof of concept (measured in reported work hours)

Based on hours recorded, we could see the share between gathering the domain specific knowledge, creating the model, creating the backend, executing and analyzing the generated test cases. (Back-end refers to the scripting back-end that translates generated meta test cases to executable format). This showed that with relatively little initial work, a good modeller can produce executable tests even in our complex test environment, which includes

different protocol simulators, system configuration interfaces and automatic log analyzers. This convinced us of the possible benefits and the decision was made to continue with the next goal to gather enough information for a business case calculation.

To calculate return on investment, we started to model an existing and already tested functionality in the selected domain. The aim was to calculate the earnings both for an initial model creation and for a model update when some new functionality is added to the existing model. We were also interested in the fault finding capability of this new technology. As a result of this exercise we proved that model-based testing is less time consuming. We achieved 15% (650 hours to 550 hours) improvement during initial modelling (creating reusable assets) and 38% (278 hours to 172 hours) improvement during incremental add-on built on top of the existing assets. We also found three additional minor faults compared to previous manual tests of the same functionality. Hours compared cover all testing-related activities like documentation, execution and test result reporting.

Based on these results we were able to prove a positive business case with break-even in 1.5 years, and we received an approval to deploy the technology on a smaller scale to enable additional studies.

Deployment scope and goals

Based on our experiences in deploying new technology, we decided to deploy MBT in a smaller project. Three different areas were selected for the deployment: Functional Testing, Call State Machine Component Testing and Database Component Testing. The goals of the deployment were to confirm the pilot results in a real project environment, to confirm the business case and ROI calculation, and to show that the technology can be adapted to other development projects. Furthermore, we also wanted to create a new mode of operation and collect engineer feedback.

Results

During Call State Machine Component Testing, manual and model-based testing activities were organized to go in parallel, in order to be able to get comparable results with both technologies. The results showed that in the first phase component testing with MBT was 25% faster compared to the corresponding manual test, while in the second phase of the implementation an additional 5%-point gain was measured in terms of work hours. In code coverage analysis MBT again outperformed manual testing, resulting in 87% coverage compared to the 80.7% measured for the manual test design in the first phase. In the second phase of implementation, this was further increased to an excellent coverage level of 92%. These higher coverage levels resulted in an improvement also in fault finding: on top of the 12 faults found with manual design, MBT discovered 9 additional software defects. According to our defect escape analysis, three of these would have been probably caught only during a very late testing phase.

During Database Component Testing, the same measures resulted in similarly attractive numbers; Component testing was performed in 22% less time than previously planned with outstanding code coverage of 93% for the database and 97% for the database interface controller.

In the Functional Testing area, our focus was on how to handle the complexity of the system under test. This was successfully resolved by our modellers. To have comparable results, part of the testing team performed manual test planning and design, while the modellers worked in parallel on their system models. The final result was in accordance with the gain in other deployment areas: 20% gain in the speed of test planning and design phase. Analysis showed that 80% of the faults have been found by both methods, but the remaining 20% of the faults have been discovered only by one of the test design methods (10% only with MBT, 10% only with manual design). This also proves that the scope of manual and model-based test design is different in nature; so combining these two approaches is highly recommended for optimal results.

Effect on every-day work

We had to focus also on integrating modelling activities into our existing software lifecycle model, and processes.

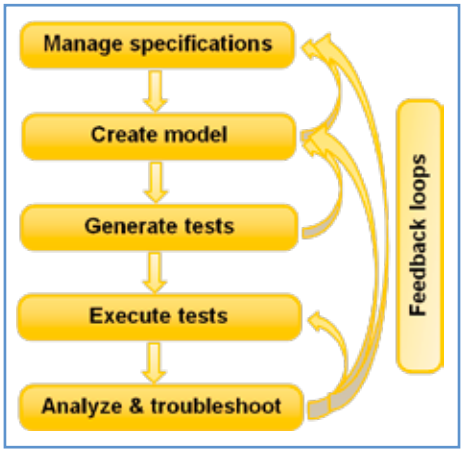


Figure 2. Feedback loops defined for all stages of modelling

To achieve this, continuous communication sessions were established in all areas and two review levels ensured feedback loops inside a sprint: a model review with modellers, specification experts, lead designers; and a test case review with all the engineers in the affected domain. Also new roles were introduced in the scrum teams: modeller and back-end scripter. An automatic documentation interface was taken into use towards our test management system to gain further efficiency.

In addition to the positive results and the changes in our daily work, the reaction of engineers was very important, so we executed a satisfaction survey to learn about the opinions of the team members. Their comments were positive: “A nicely built model can be understandable for anyone. There is no need to check several other documents, only the model.” “Modelling is fast and comfortable after having some experience.” “It is also motivating that serious code defects were found with MBT (even after executing manually designed test cases).”

Lessons learned

In the nearly 3 years we’ve put in for introducing this new technology, we have learnt that MBT can be successfully adopted on complex projects and can introduce significant improvements to testing. However, to achieve this, a step-by-step pilot and deployment approach is needed with clear goals, Go/No-Go decision criteria, and business sponsorship. Strict reporting practices

are needed for business case validation to show fault finding and work effort improvements. Continuous follow-up for proper ramp-up with on-site and remote support from the vendor will help in initial problem solving and give guidance during the first model reviews. It was also important to not train only the modellers, but some key technical stakeholders of the project as well, like specification experts, lead designers and test architects, and some sponsors from management.

As Pertti Lukander, head of industry environment at Nokia Siemens Networks said in the newsletter of ETSI (24 October 2011) on Industrial Deployment of Model-Based Testing: "Our deployment results proved that Model-Based Testing (MBT) is the right answer to many challenges in software testing. At Nokia Siemens Networks, we are convinced that we can serve our customers better with this technology."

> biography

Gábor Federics
is test manager at Nokia Siemens Networks responsible for model-based testing deployment for a GSM-R project. He joined Siemens after graduating as an electrical engineer in 2004 and later moved to Nokia and Nokia Siemens Networks. He has held various positions in software testing in all three companies.

Szilárd Széll
is business improvement manager at Nokia Siemens Networks with focus on agile transformation. He has almost 12 years of experience in testing and change management. Szilard joined Nokia as test engineer in 2000 and has since been working as test team leader and scrum master both at Nokia and Nokia Siemens Networks. He has a degree in informatics and mathematics, as well as IT management, and holds CSM and ISTQB AL-TM certificates.

Project Management And Its Importance

by Santosh Varma

What is project management ?

In simple words, project management would be accomplishing a project within a given scope, time and cost. As the definition of project stands – a temporary endeavor to create unique goals and objectives within a given cost, time and scope. This is in total contrast to the definitions for program and operations. Operations are permanent in nature which means that they are ongoing processes with no definite timeframes.

We as human beings are following project management in day-to-day life. At home also, we need to define the scope, time and cost for every activity we plan. For all of the activities we plan to undertake, there is initiating, planning, executing, monitoring and controlling, and finally the closure of the activity.

In practice, project and operations are two separate systems and require management expertise to tackle any issues which arise.

There are various approaches to managing project activities which include: Agile, incremental, interactive and phased. Irrespective of the approach used and implemented, careful consideration must be given to the overall project objectives, timeline and cost as well as to the roles of the stakeholders.

Why project management tasks ?

Basically, this is a question which I have often come across, and people ask me about the relevance of the project management tasks and activities. In my opinion, there are the following tasks which the project manager undertakes during the entire project life cycle:

- Status reporting
- Schedule management
- Co-ordination among team members (including stakeholders)
- Project budget
- Conducting regular project status meetings

Projects can succeed without following any project management tasks/activities, but the long term effects would be disastrous. For example: The status report is what drives the formal communication between the delivery team and the customer team. If status reporting is not applied and followed, then there is no official record of what was agreed between the customer and the delivery team, and the delivered project/product may not match the customer's requirement. Similarly, all the stakeholders need to be kept informed from the beginning of the project so that all the needs/requirements are taken care of. However, in case we do not maintain stakeholder registers, the team will not be able to tell at a particular point of time what a particular stakeholder requested and who the relevant stakeholder is.

If we think of a scenario without project manager and project management activities, that can only be imagined for small engagements. My opinion in this respect is that some formal project management and practices should be in place for a project to be successful. Customers expect satisfaction, and they expect timely and efficient systems to be delivered and this cannot happen without PROJECT MANAGEMENT.

Type of project life cycle to be selected: Agile or Waterfall

While there are probably as many unique project management approaches as there are project managers, there are two well-known production cycle methodologies that have been the topic of much discussion in PM circles, i.e. the Agile and the Waterfall methodologies.

In my opinion, the selection of the life cycle for a particular project depends on how clear the requirements are to the project team. According to my experience, it is better to use Waterfall, because Waterfall projects assume all (or most) requirements which can be understood before designs are created, and that coding follows both. This assumes a high degree of certainty regarding the requirements and fewer errors, whereas Agile assumes that the requirements are not or cannot be known until a working version of the system is created. This will have significant implications for cost and time estimation afterwards.

And secondly, if the client wants a release of software on a periodic basis, then Agile might be the best methodology to be adopted.

As I evolve in my own area of expertise, I am trying to constantly reinvent small aspects of what I consider best practice. Most recently, to address the incredibly complex requirements of a large client initiative, maybe the combination of Waterfall and Agile approaches and principles should give us the required and superior results.

Stakeholders:

Project management's mantra for success is: *"Satisfy your Stakeholders"*; and once this satisfaction has been granted to the respective stakeholders we can rest assured that the project will be successful and all the requirements will be met.

By stakeholders we mean any person who positively or negatively influences the outcome of the project. Examples of stakeholders include: customer, user group, project manager, development team, testers etc. Stakeholders are anyone who is interested in the project. They may also exert influence on the project's outcome and objectives. The project management team must identify these stakeholders, determine their requirements and expectations and to the extent possible manage their influence in relation to the requirements to ensure that a successful project is delivered.

Identification of these stakeholders should begin as early as possible in the project because the earlier the requirements are gathered from these stakeholders, the better it would be for the project in terms of scope, time and cost.

Estimation:

In project management, accurate estimation is the backbone of sound project planning. Without an estimation, the project will be carrying on infinitely, thus impacting the constraints of the projects, namely scope, time and cost. Many processes have been developed to aid engineers in making accurate estimations, such as analogy based estimation, parametric estimation, Delphi method, function points, and Program Evaluation and Review Technique (PERT).

A sound estimate starts with the creation of a WBS (Work Breakdown Structure). A WBS is a list of tasks that – if completed – will produce the final product and deliverable. Once the WBS is created, the team must create an estimate of the effort required to perform each task. The most accurate estimates are those which rely on previous experiences (analogous estimation). Team members should review previous project results and find how long similar tasks in previous projects took to complete. Causes of delays in the past should be taken into account when making current estimates.

No estimate is guaranteed to be accurate. There can be various scenarios, e.g., people get sick, leave the organization, teams run into unexpected technical issues and problems, etc. The main objective of estimation is not to predict the future, instead the main objective is to create an honest, well informed opinion of the effort required to perform a particular task.

In my opinion, a project manager can help the team create accurate estimates by reducing uncertainties about the project. The most effective and productive way to do this can be:

- Creating an accurate and detailed scope document
- Reaching a consensus on the tasks that have to be performed
- Discussions on assumptions with the entire team

Risk management:

There are nine knowledge areas in project management, which are: integration, scope, time, cost, quality, risk, communication, HR and procurement. To my view, the most important ones which affect project on a very large scale are: scope, time, cost, risk and quality. I do not wish to under-estimate communication, but if feel that risk management also plays a valuable role in project management. The earlier the risks are identified, the better it is for the project and the team to succeed. Risks can come from any of the following:

- Project Failures
- Natural causes and disasters
- Deliberate attach from adversary
- Uncertain financial markets
- Other uncertain events

Once the risks have been identified, we need to prepare strategies to manage risks. The strategies include:

- Transfer risk
- Avoid risk
- Accept risk
- Reducing the negative effect or probability of the risk

My experiences in project management activities:

From my personal experiences, project management is a wonderful experience.. In my opinion, the most important trait for a project manager has to be TRANSPARENCY. If he/she is open, then other factors (scope management, time management, risk management, cost management, etc.) will automatically fall in place. Other traits that follow transparency can be: project management experience, communication, guidance and mentoring, and motivating the team.

The entire team looks to the project manager for motivation and leadership, and if the project manager stands up and delivers, then the entire team will support the project manager which in turn benefits both the project and the organization.

Challenges in project management:

In my opinion, some of the challenges faced while undertaking project management activities include:

- Lack of project management experience
- Time, cost, schedule
- Resource handling
- Conflict handling
- Risk handling
- Unrealistic deadlines
- Communication
- Scope changes
- Uncertain dependencies
- Insufficient team skills
- Lack of discipline and accountability

- Non-involvement of customers and end-users during the project
- Vision and goals not well-defined
- Shifting organizational priorities
- Change control not well-defined
- Geographically dispersed project teams
- Using the wrong tool for the job
- Spending too much time in project/status meetings

And ... not to forget, last but not least – **QUALITY**!

Project leadership is a skill that takes time to develop in a person or organization. Achieving success requires analyzing setbacks and failures in order to improve. Focusing on each project's challenges and learning from them will help to build a more capable and successful project management capability.

Conclusion:

Project management activities, in some form or other, have been used for more than 5,000 years, yet there is still more to learn in the continual development of the techniques, practices, and procedures of modern management skills. In the past 50 years or so, the use of project management has grown exponentially, which can only be attributed to the sharing of information. Many books have been published and people are getting trained in project management, but still there are gaps in practice and performance that can only be filled through sharing of personal experiences with project management competence.

Sharing is easy when one can mentor or write a short article on project management practice. Internal project electronic bulletin boards may be used to inform the team. If one has something to share, we should find a means to get it to as many people as possible.

> biography

Santosh Varma

I am Santosh Varma, IT professional and working in this industry for the past 14 years. I am currently living in Bangalore, India, where I work as Test Manager in an IT company and also undertake some project management activities.

Can agile be certified?

Find out what Aitor, Erik or Nitin think about the certification at **www.agile-tester.org**

Training Concept

All Days: Daily Scrum and Soft Skills Assessment

Day 1: History and Terminology: Agile Manifesto, Principles and Methods

Day 2: Planning and Requirements

Day 3: Testing and Retrospectives

Day 4: Test Driven Development, Test Automation and Non-Functional

Day 5: Practical Assessment and Written Exam

27 training providers worldwide!

Certified Agile Tester

We are well aware that agile team members shy away from standardized trainings and exams as they seem to be opposing the agile philosophy. However, agile projects are no free agents; they need structure and discipline as well as a common language and methods. Since the individuals in a team are the key element of agile projects, they heavily rely on a consensus on their daily work methods to be successful.

All the above was considered during the long and careful process of developing a certification framework that is agile and not static. The exam to certify the tester also had to capture the essential skills for agile cooperation. Hence a whole new approach was developed together with the experienced input of a number of renowned industry partners.

Supported by

Barclays
DORMA
Hewlett Packard
IBM
IVV
Logic Studio
Microfocus
Microsoft
Mobile.de
Nokia
NTS
Océ
SAP
Sogeti
SWIFT
T-Systems Multimedia Solutions
XING
Zurich

Mistake metamorphism

by Trinadh Kumar Bonam

This article will be useful for those who are looking for a clear difference between the different terminologies that are used for the term 'defect' in the SDLC.

The main intention of this article is to bring a clear picture about the 'mistake' committed by an analyst in the early stages of the SDLC, which leads to a 'defect' in the final stage of the SDLC. I would also suggest using the relevant term in the respective phase of the SDLC, which would be beneficial for process improvement and for calculating metrics.

In this context, the concept of 'Mistake Metamorphism' is explained by taking the definitions given by IEEE and ISO and by streamlining them by mapping them to the respective phases of the SDLC.

Introduction

In software development projects, a 'mistake' or 'fault' can be committed at any stage during development. For each phase, we have different techniques for detecting and eliminating the mistake/faults that originate in that phase. However, no technique is perfect, and it is expected that some of the mistakes/faults of the earlier phases will finally manifest themselves in the code. This is particularly true because in the earlier phases most of the verification techniques are manual as long as no executable code exists. Ultimately, these remaining mistakes/faults will be reflected in the code and continue into the subsequent phases of the SDLC.

Why does software have defects? The reasons include:

- Miscommunication
- Inappropriate skills
- Complex technology
- Programming errors
- Tight schedules
- Poorly documented code
- Changing requirements
- Failure to communicate and act as a team
- Lack of domain skills

When do software defects occur?

- The software doesn't do something that the specification says it should do
- The software does something that the specification says it shouldn't do
- The software does something that the specification doesn't mention

The software doesn't do something that the specification doesn't mention but should; the software is difficult to understand, hard to use, slow, or – in the software tester's eyes – will be viewed by the end user as just plainly not right.

Mistake Metamorphism:

The word "metamorphism" comes from Greek, whereby meta = change, morph = form, so metamorphism means to change form. We can use this word in software engineering for the evolution of a defect in the final stage of software deployment. The main intention is to show a clear picture about the 'mistake' committed by an analyst in the early stages of the SDLC, which leads to a 'defect' in the final stage of the SDLC.

Different stages of a 'mistake' in the entire SDLC can be described by using the following terms:

- Mistake
- Anomaly
- Fault
- Failure
- Error
- Exception
- Crash
- Bug
- Defect
- Incident
- Side effect

You might be amazed that so many names could be used to describe a software failure. Why so many? If you look up these words in the English Dictionary, you'll find that they all have slightly different meanings.

But in the virtual software dictionary, the name of a software failure will be changed based on the phase in the SDLC; this is what we are going to prove.

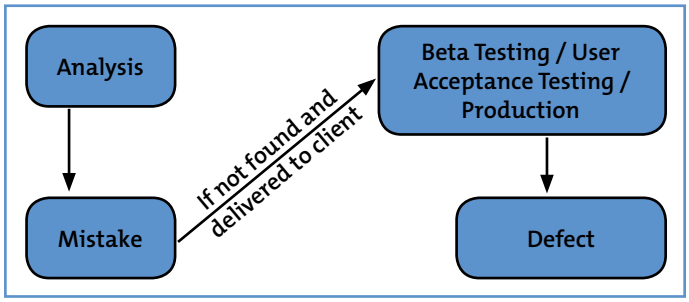


figure 01

The high level view (figure 01) tells how a 'mistake' committed by an analyst leads to a 'defect' in the software product/project, once it has been delivered to the client. We will call this a 'Mistake Metamorphism'.

Let's see the example below which describes in more detail how this 'mistake' committed by an Analyst in the initial stage of the SDLC leads to a 'Defect' in the software project/product.

The diagram (figure 02) depict the transformation of a 'Mistake' to a 'Defect'

An Analyst commits a 'mistake' which leads to an anomaly:

a) The customer explained the requirement(s) and gave business requirements documents (BRDs) to the analysis team.
b) In the requirements phase of the project,the analyst(or analysts) will prepare a functional specification from the customer's BRDs. At this time they may make mistakes in any of the following ways:

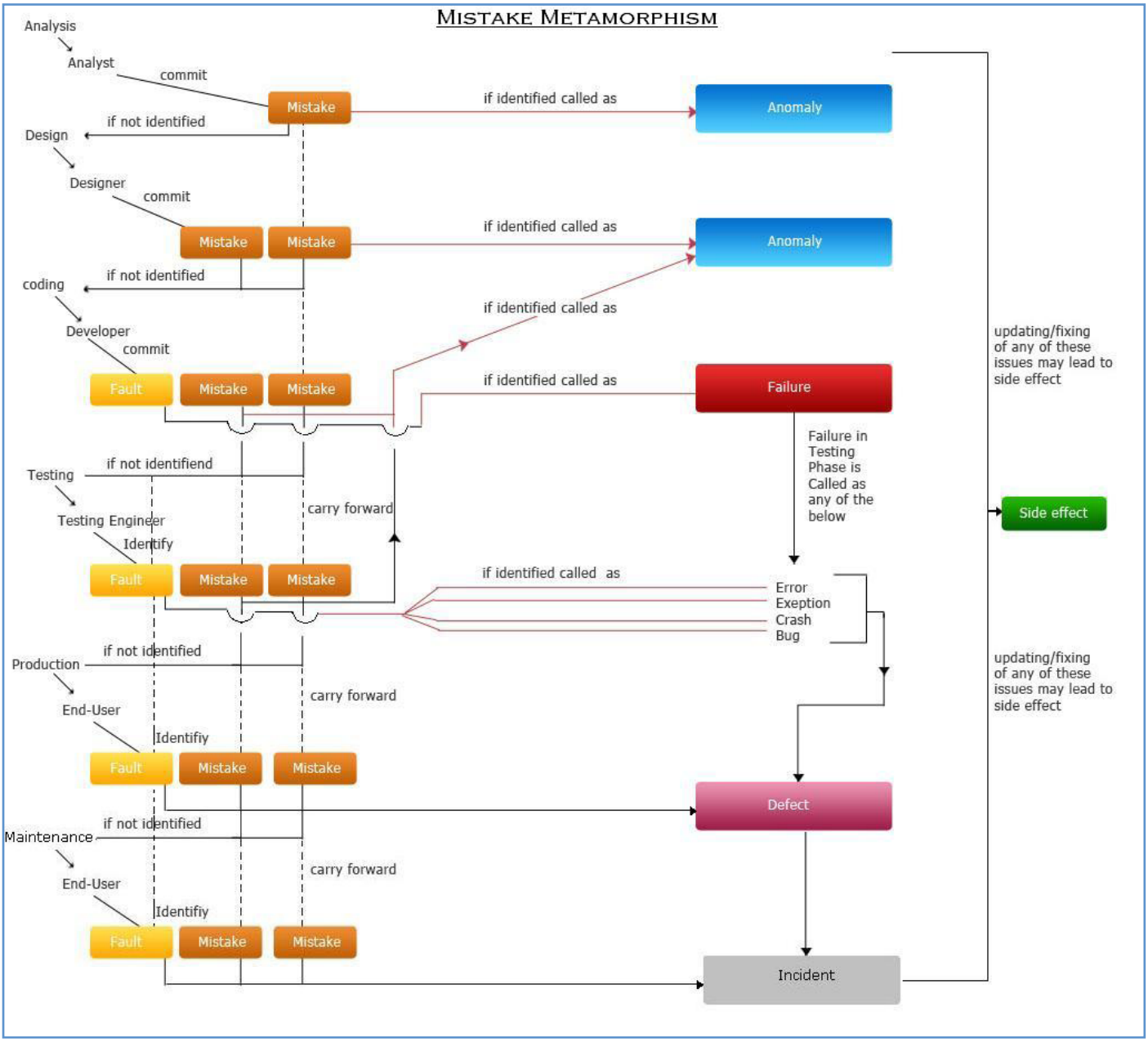


figure 02

i) Misunderstanding the business requirement
ii) Mistakes in the documentation
iii) Inconsistency in the functional specification

Let's correlate the above points (a) and (b) with the definition that is provided by IEEE for the term 'mistake'.

Mistake (definition by IEEE):

A human action that produces an incorrect result, and the subject of the quality improvement process.

The information provided as examples in (a) and (b) precisely reflect the IEEE definition of 'mistake'.

c) If the 'mistake' committed in the functional specification document is identified by the same analyst, another analyst, a designer, developer or a test engineer, then this mistake can be called an 'anomaly'.

Let's correlate the above point (c) with the definition that is provided by the IEEE for the term 'anomaly'.

Anomaly (definition by IEEE):

Anything observed in the documentation or operation of software that deviates from expectations based on previously verified software products or reference documents.

The information provided in point (c) precisely reflects the IEEE definition of 'anomaly'.

A designer commits a 'mistake' which leads to an anomaly:

d) As part of the design activities in the design phase, software designers will prepare system design and detailed design documents. If the 'anomaly' is not caught by any of the people in the project during the design phase, the same 'mistake' will be carried forward.

A developer commits a 'fault' or 'mistake' which leads to failure:

e) In the coding phase, developers write the code as per the design documents. Developers may make any of the following mistakes:

i) There may be typing errors
ii) Incorrect logic
iii) Cross-Site Scripting
iv) Choosing the wrong database, etc.

A mistake committed by the developer in the code can be called a 'fault'.

Let's correlate point (e) with the definition that is provided by the IEEE for the term 'fault'.

Fault (definition by IEEE):

An incorrect step, process, or data definition in a computer program which causes the program to perform in an unintended or unanticipated manner.

The information provided in point (e) precisely reflects the IEEE definition of 'fault'.

Once the coding has been completed, the code will be reviewed/tested as part of unit testing. If the 'fault' is identified in unit testing, then that 'fault' is called a 'failure'.

Let's correlate the same with the definition that is provided by the IEEE for the term 'failure'.

Failure (definition by IEEE):

The inability of a system or component to perform its required functions within specified performance requirements.

A 'fault' identified as part of a review or unit testing exactly resembles the definition that is provided by the IEEE for 'failure'.

Any faults that are not identified in unit testing will be carried forward and can then be identified in integration or system testing. At this stage, the identified fault(s) is/are called

6. Error
7. Exception
8. Crash
9. Bug

Let's first discuss the above points one by one.

1. **Error:** Errors type are:

- Run-time error,
- Syntax error, and/or
- Semantic error.

These errors will be thrown by the system if there is discrepancy in the code.

Run-time error, semantic error – an error in logic or arithmetic that must be detected at run time.

Syntax error – an error of language resulting from code that does not conform to the syntax of the programming language; "syntax errors can be recognized at compilation time"; "a common syntax error is to omit a parenthesis".

Let's look at the definitions provided by ISO.

Error

1) (ISO) a discrepancy between a computed, observed, or measured value or condition and the true, specified, or theoretically correct value or condition;
2) A quality problem discovered by software engineers (or others) before the software is released to the end-user (or to another activity in the software process). [Pressman, 203]

The above definitions exactly resemble the types of error that are identified in system/integration testing.

1. **Exception:** There are some faults that occur in rare conditions because of problems in hardware, impact of power, unintentional memory consumption (i.e., memory leaks). These types of faults are called 'exceptions'.

Díaz Hilterscheid

Online Training

ISTQB® Certified Tester Foundation Level (English & German)
ISTQB® Certified Tester Advanced Level – Test Manager (English)
ISTQB® Certified Tester Advanced Level – Test Analyst (English)
ISTQB® Certified Tester Advanced Level – Technical Test Analyst (English)
ISEB Intermediate Certificate in Software Testing (English)

iseb Qualifications from BCS

ISTQB® Accredited Training Provider

ISTQB® International Software Testing Qualifications Board

Díaz Hilterscheid

te testing experience

Note: Memory Leak: A bug in a program that prevents it from freeing up memory that it no longer needs. As a result, the program grabs more and more memory until it finally crashes because there is no more memory left.

Here is the definition that is provided by the IEEE for 'exception', let's correlate the same with the above cited point.

Exception (definition by IEEE):

An event that causes suspension of normal program execution; types include addressing exception, data exception, operation exception, overflow exception, protection exception, and underflow exception.

The above definition exactly resembles the 'exception' that is identified in system/integration testing.

2. **Crash:** All the above types of faults (i.e., failure, error and exception) do not impact the software that much. There will be almost no any data loss or software crash. A fault that is caused because of fire, accident, natural disaster, etc. will, however, lead to a complete failure of the software. In most cases, the system cannot be recovered. This type of failure is called a 'crash'.

Here is the definition of 'crash' given by the IEEE:

Crash (definition by IEEE):

The sudden and complete failure of a computer system or component.

Let's elaborate point 4, i.e., 'bug' in more detail.

A tester might miss the 'fault' or 'mistake':

j) If there is a fault in the code that may not be thrown as an error to the user, then this type of fault is called a bug. Bugs can be

   i) Programming errors,
   ii) Computer error, and/or
   iii) Functional cohesion.

i) Programming error: An error resulting from bad code in some program involved in producing the erroneous result.
ii) Computer error, error (computer science): The occurrence of an incorrect result produced by a computer
iii) Functional cohesion: A type of cohesion in which the tasks performed by a software module all contribute to the performance of a single function.

These bugs will be identified by the test team during integration or system testing.

Let's look at the definition for the term 'bug' provided by the IEEE.

Bug (definition by IEEE):

(1) Things the software does that it is not supposed to do, [or] something the software doesn't do that it is supposed to. [Telles and Hsieh, 5];

(2) A fault in a program which causes the program to perform in an unintended or unanticipated manner.

Note: This term is one of the oldest in the software profession. It originated in the 1950's when the late Admiral Grace Hopper found an actual insect hamming the contacts of an electromechanical switch. The term now refers to errors or defects that find their way into programs and systems. Some software quality researchers find it desirable to replace the pejorative term "bugs" with other terms such as errors, defects, faults, and failures.

The above definition exactly resembles the types of faults that are identified in system/integration testing.

A missed 'error', 'exception', 'crash', or 'bug' in testing will lead to a 'defect' in production:

k) If the faults are not identified prior to the software release, then they are identified once the software has been released to customer after its deployment in the production environment. If the faults are identified during beta-testing, or user acceptance testing, or in production *(here support will be considered for a certain period based on the contract)*, then these faults are called 'defects'.

Here are the different definitions that are provided for the term 'defect':

Defect

1) (IEEE) a product anomaly;
2) Synonymous with fault, both imply a quality problem discovered after the software has been released to end-users (or to another activity in the software process); [Pressman, 203]
3) fault; [SWEBOK, Ch.5, p.6]
4) error, fault, or failure; [SWEBOK, Ch.11, p.9]
5) non-conformance to requirements.

As we are trying to prove that #the terms for software failure will change based on the phase in the SDLC, in the virtual software dictionary, let's consider the definition provided by Pressman for 'defect'.

The above definition exactly resembles the fault(s) that is/are identified in post deployment.

Though the maintenance/sustenance phase is segregated, we are aware of the fact that once the software has been deployed, it will be in the maintenance phase. During this phase there won't normally be any deployments or upgrades to the software, except for hot fixes for the defects that are identified in the maintenance/sustenance phase. Here complete support will be provided by the team that developed the software or by the vendor who took it over from the development team/vendor. Any issue that is identified in the maintenance/sustenance phase can be called an 'incident', which may or may not be a problem/issue. This is called a 'ticket' by most people. The issue could be due to a lack of user training on the software, to environmental issues or due to non- reproducible issues or some other software compatibility issues, etc.

Here is the definition that is provided by the IEEE for the term 'incident'.

Sie suchen
das Besondere?

Díaz Hilterscheid

Incident (definition by IEEE):

An incident is an event occurring during testing that requires attention (investigation). [IEEE 1008] Incidents do not necessarily justify the filing of a formal problem report.

The definition of the IEEE exactly resembles the occurrence of issues in the maintenance phase.

Issues that are identified and fixed in production (during the maintenance phase) lead to 'side effects':

l) Updating/fixing any of the issues identified in the maintenance phase (i.e., anomaly, failure, error, exception, crash, bug, defect) may impact other areas of the updated functionality/code, which leads to a new issue called a 'side effect'.

Side effect (definition by IEEE):

An unintended alteration of a program's behavior caused by a change in one part of the program, without taking into account the effect the change has on another part of the program.

(...)

References:

Standards:

IEEE Standard 610.12-1990. IEEE Standard Glossary for Software Engineering Terminology.

IEEE Trial Draft SWEBOK Version 1.00. Software Engineering Body of Knowledge.

Books:

Pressman, Roger S. Software Engineering: A Practitioner's Approach. 5th ed. New York: McGraw-Hill, 2001.

Pankej Jalote An Integrated Approach to Software Engineering, 2nd Edition.

> biography

Trinadh Kumar Bonam

Mr. Trinadh Kumar Bonam, MSc (Computer Science), MTech (Software Engineering), Six Sigma (GB), Certified Scrum Master, HNC (from NIIT), ISTQB® foundation, has worked as test engineer and as test lead. He has seven years of testing experience in domains like sales/partner sales, call center, telecommunications (OSS/BSS/VAS), retail, manufacturing, etc. As a Test Analyst, he has worked in various projects covering multiple technologies (like Siebel, MS CRM, etc.) and has been involved in agile testing.
He now works for the established No. 1 company in India. His primary objective is ‚QUALITY'. He is interested in learning new things, he finds never ending challenges in testing. He can be reached at trinadh.bonam@gmail.com, and you can find more about him on LinkedIn.

Wir auch!

Díaz Hilterscheid

Lassen Sie sich anstecken von der kollegialen Arbeitsatmosphäre in einem starken und motivierten Team. Zum nächstmöglichen Termin stellen wir ein:

Senior Consultants
IT Management & Quality Services (m/w) für SAP-Anwendungen
Deutschland und Europa

Sie haben

- eine fundierte Ausbildung oder ein Studium (z. B. Informatik, BWL, Mathematik) sowie mehrjährige Berufserfahrung als IT-Consultant in einem Fachbereich bzw. in der Organisation eines SAP-Anwenders, eines IT-Dienstleisters oder in einem Beratungsunternehmen in entsprechenden Projekten
- ausgewiesene SAP-Kenntnisse (z. B. SAP ERP, SAP BI oder SAP Solution Manager)
- Erfahrung im Customizing und mit ABAP-Programmierung
- Kenntnisse in der praktischen Anwendung von Methoden und Standards, wie CMMI®, SPICE, ITIL®, TPI®, TMMI®, IEEE, ISO 9126
- Erfahrung in der Führung von großen Teams (als Projektleiter, Teilprojektleiter, Testmanager)
- Vertriebserfahrung und Sie erkennen innovative Vertriebsansätze

Sie verfügen über

- Eigeninitiative und repräsentatives Auftreten
- eine hohe Reisebereitschaft
- gute Englischkenntnisse in Wort und Schrift

Dann sprechen Sie uns an – wir freuen uns auf Sie!

Bitte senden Sie Ihre aussagekräftige Online-Bewerbung an unseren Bereich Personal (hr@diazhilterscheid.de). Ihre Fragen im Vorfeld beantworten wir gerne (+49 (0)30 74 76 28 0).

Díaz & Hilterscheid Unternehmensberatung GmbH
Kurfürstendamm 179, D-10707 Berlin
www.diazhilterscheid.com

Mehr Stellenangebote finden Sie auf unserer Webseite:

- Senior Consultants Financial Services (m/w)
- Senior Consultants IT Management & Quality Services (m/w)
- Senior Consultants IT Management & Quality Services (m/w) für Agile Softwareentwicklung und Softwaretest
- Senior Consultants IT Management & Quality Services (m/w) für unsere Kunden aus der Finanzwirtschaft

Traditional vs Agile methodologies

by Leanne Howard

How do you collect the metrics to allow you to pick the most appropriate methodology to fit your context?

This article outlines a study conducted to compare different methodologies using a controlled environment, the same test input documentation and similarly qualified professional testers. Whilst it is a fairly small project, the results are striking, and contained within this article some conclusions have been suggested. We also threw in a change of tool and analysed this impact as well!

The methodologies compared were:

- Waterfall
- Agile
- Bug Hunt

Starting Point

In order to be able to provide and put forward valid ideas, as clinical an initial environment as possible needed to be designed, to be the starting point.

This was –

A Business Requirement document (BRD) which was:

- as close to real life as possible
- a mix of actual business requirements and some functional specifications
- written as paragraphs of text, not individually identified requirements, which then needed to be extracted by the test team
- not given priority to the individual requirements by the Business
- contained some prototypes for some of the screens

The testing team were all test professionals

- Test managers were all ISQTB® certified Advanced Test Managers
- Test analysts/senior test analysts were at least ISQTB® certified at Foundation level and the seniors at Advanced Test Analyst level
- those undertaking the agile methods were Certified Agile Testers (CAT)

Application

- each team had access to the same application on which they could complete their execution
- the application was in the banking domain in which most staff had some previous experience
- the application was seeded with defects

Tools

- JIRA was used for the traditional methodologies with Planit customised templates to record requirements, test conditions, test cases and defects
- JIRA was used for one team using Agile with Planit customised templates for user stories, session sheets and defects
- HP Accelerator was used for one team using Agile with Planit customised templates for user stories, session sheets and defects (not out of the box functionality)
- The Bug Hunts used Planit customised templates simply using Microsoft Word and Excel

Team

- Team 1 – Agile using JIRA
- Team 2 – Bug Hunt 1
- Team 3 – Bug Hunt 2
- Team 4 – Traditional (Waterfall)
- Team 5 – Agile using QC Accelerator

It should be noted that no member of staff was included within more than one team as part in this study. Also, only team members were given access to the tools that they were using and the project data was locked down. Staff were asked not to discuss what happened within their teams and how many defects or the amount of coverage they were able to achieve outside of their team.

Metrics

There were a number of metrics gathered:

- Defects split by total number and severity
- Coverage of requirements

Also observation techniques were used to gather data which is used in this article.

Findings

There were a number of findings and these have been split into the following sections to highlight certain aspects.

Defects

Note: Where defects are quoted in the article, it only includes high severity ones. All others have been ignored for the purposes of this article as the study parameters were to focus on high severity defects. A full analysis of all defects found may be a subject of a further article.

Teams	Total Defects	High Defects Severity
Team 1	105	**29**
Team 2	52	11
Team 3	65	15
Team 4	99	25
Team 5	45	15

All defects discovered were required to be logged, as the scenario was that the development team was remote to the testers. Whilst this was not the ideal environment for the Agile teams and could have impacted on the productivity, we needed a way to further analysis the quality of defects found.

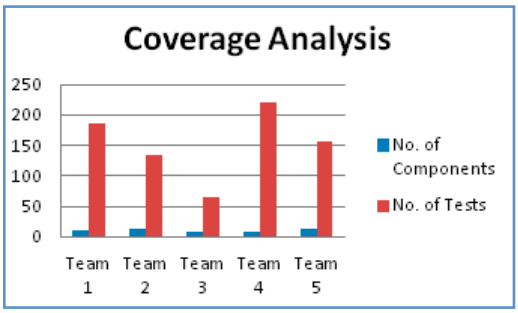
The Agile team found the most defects and the greatest number of high severity defects of all the teams. It was interesting to discover that there was a divergence in the type of some of the defects found, with the Agile team more focused on usability defects whilst the traditional found more in the database.

Teams	% High Severity Defects
Team 1	28%
Team 2	21%
Team 3	23%
Team 4	25%
Team 5	33%

Another discovery was that both the Agile teams scored the highest percentage of high severity defects when compared to the total number of defects that were found by that team, which therefore indicates that Agile finds the higher percentage of high severity defects. This would be an indicator of the quality of testing when using an agile methodology.

When comparing the defect leakage, Team 4 came out best; however, Team 1 were only 6% lower and Team 3 was next which differed by another 4%. When you compare the number of man days to get an additional 6% or so, you have to question whether this is value for money.

Test Coverage



The traditional team was told to perform system testing only, although was not given any boundaries around functional or non functional testing. The team did, however, define the scope of testing in their Test Plan to exclude any non functional testing and to be fair there were few non functional requirements contained within the BRD and they did not have the tools available to do this testing. They did, however, do testing around the database, using SQL queries, and the tests were written to a much more technical level. Perhaps an interesting further study would be to create teams including non functional tests and compare those.

The Agile teams of course tested to the acceptance criteria, as demonstrated to the PO as part of the iteration review meeting, and was therefore much more customer focused, so they did little database testing and Team 1 was the only team to perform usability testing.

There were 12 major components to provide full coverage and most of the teams did recognise these, although they did define them slightly differently. There was a high variance in the number of tests that each team executed.

It should be noted that the two Agile teams looked at coverage slightly differently, in that Team 1 were much more collaborative and due to their numerous inter team discussions found that the system interaction for the same function was slightly different between the team members and therefore increased the usability testing. Team 5 focused only on the acceptance criteria where usability was not specifically defined as a criteria or discussed with the Product Owner.

Test Efficiency

Defects found per man day showed that Team 1 was slightly more efficient than Team 3 with Team 2 a close third. The man days were taken over the whole period of the project, as defects can be found in the preparation phases of the project as well.

The teams basically covered the same areas of testing although they split them up slightly differently. There was quite a large variation in the number of tests each team completed in order to represent this coverage. There were also differences in the splits between planning, preparation, execution and closure activities across each team. Certainly the teams with less ceremony (process, documentation etc.) did not decrease the coverage and in some cases spent more time testing and therefore finding defects.

Teams	Defects per man day
Team 1	1.26
Team 2	0.91
Team 3	1.25
Team 4	0.47
Team 5	0.6

Agile Experience

Our starting state was that staff working on the Agile projects were Certified Agile Testers (CAT). This meant that there was no time allowance for learning a new methodology and the teams were able to get straight into Release/Iteration planning. They followed the Agile process as defined within the CAT course (details can be found at www.planit.net.au). Team 1 were more experienced in Agile and had been involved in previous Agile projects. It was marked that they had also changed their mindset to a more collaborative way of working and the personalities worked well together.

Team 5 tended to still revert back to some of the traditional principles and there were fewer conversations within the team. The Agile principles had not yet been embedded into their way of working. This team would have benefited from some additional coaching which was not given due to the study. This indicated that newly formed teams may require some initial support in order to bed in the practices, which is often not factored into projects. It is all too easy when put under pressure, whether that be real pressure or self imposed, to revert back to what we know rather than what we have just been taught.

Product Owner Involvement

The Product Owner (PO) or Business representative was available for all teams, although throughout the study we simulated real life as much as possible in that the teams had to pre-book meetings with their Business representative (PO) and they were not always available to answer questions instantly. The PO contributed and confirmed the acceptance criteria with both Agile teams which then directed the scope of testing.

The exception to this was the two Bug Hunt teams. The process for them was that the Business representative had an initial scoping meeting with the whole team and then there was no contact again until the Test Summary / Recommendation report at the end of the process. These teams relied on past experience, domain knowledge and defect taxonomies for testing along with the BRD and the initial meeting. This is a short timeboxed activity, so relied much more on the quality of the staff rather than the external inputs.

For the traditional team they compiled a list of questions following a formal review of the requirements which they then walked through with the Business representative. The Business also had opportunity to review both the test conditions and test cases and provide feedback before sign-off. Therefore there was plenty of opportunity for the Business to confirm the coverage was appropriate or agreed with the team to add more tests.

In the beginning Team 1 had more opportunity to interact with the PO which they took advantage of; this meant that the acceptance criteria were clearly defined. Team 5, however, did not get so much initial time with the PO due to his time pressures and this contact did not really increase although they had the opportunity to do so. This did have a significant impact on the quality of the deliverable and in particular the number of defects found by this team.

Collaboration

It was observed that the teams that collaborated well together produced the highest quality of testing both in the form of coverage and the number of defects. Interestingly, this seems to be irrespective of the methodology. There were teams which did this better than others and it seemed to have more to do with the individuals than the methodology they were using.

One interesting fact was the two Agile teams, which one would think would collaborate more closely, were sat in two differently configured rooms. They were in separate rooms where only the team members were sat in that room. One team had a set of desks in the middle of the room facing each other, while the other team had their desks in a "U" shape pushed against the walls, so in effect they had their backs to each other when they were working. This second environment did not promote constant interaction and it was apparent in the results that the team achieved, which were not as good as the other team's. This is something that we are going to explore further, as it was not one of the set metrics we had thought to measure beforehand.

Type of person

One of the greatest findings from the study was the fact that, if you have the right people on the project there is a greater likelihood of a successful outcome. It is hard to define what the "right" person is, as it will depend on the context of the project and the environment in which they work.

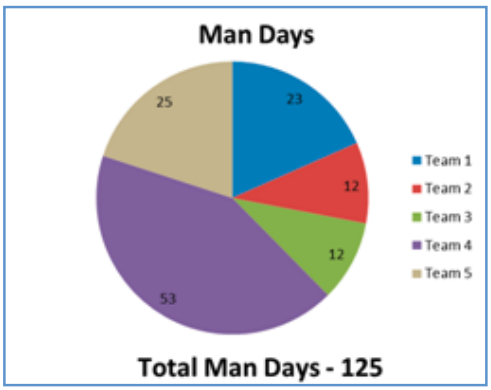
However, here are some suggestions of what to look for:

- Passion and enthusiasm for testing
- "Can do" attitude
- Teamwork mentality, working for the good of the team not the individual
- Excellent communication skills, high level of interaction
- Common-sense in applying the correct testing techniques based on project drivers

We certainly found that those teams that had the greater collaboration within the team had the best success rate. This has highlighted that communication skills and the ability to work well within a team have a significant impact on the quality. This finding held true irrespective of the method used and is something that recruiting managers need to embrace and work out how to assess as part of job interviews.

Another conclusion that we were able to draw was that team leadership seemed to play a large part in accomplishing the objectives. I want to make it clear that I am talking about leadership and not management, which are vastly different. In Agile the leader comes from within the team and this can vary from person to person as the project progresses. Team 1 certainly shared this responsibility and it seemed to shift to whomever had the best knowledge within the area under test or discussion that was happening. This was less evident in Team 5 and leadership responsibilities seemed to stay with the more experienced member of the team.

Length of the project



Again, this was not one of the facts that we set out to discover. However, it did seem to make an impact all be it that, it may have been on the "feel good" factor of the people in the team and their enjoyment of the project experience as a whole, rather than directly on the overall project quality. This is important because if your staff are happy, they are more willing to commit to the project, even in pressured times. Of course, this is only one factor.

The Bug Hunts are a set number of days where the key objective is to find as many high severity defects as possible. As there is not a great deal of interaction with the Business stakeholder, the premise is that the testers rely on their experience in the domain or general testing experience. It is likely that staff used defect taxonomies even though they were not specifically written down. The high energy and concentration was focused for this short period of time.

The Agile teams, whilst they had not been set timescales, had an expectation from the staff that they would be fast paced, but with a relatively short timeframe. Although, one of the teams let this drive them, rather than the stated objective of quality, which in fact was borne out by their results as they found less defects.

The traditional project knew that they were going to be assigned to this project for a longer period of time which was based on their past experience. Whilst this did not reflect in their enthusiasm or productivity, there were some interesting reactions when they learnt that the Agile team, which had started at the same time, had completed their project and the traditional team was not even half way through, in fact they had not even touched the system or started execution at this time.

Project drivers

What part does quality play – is the driver to get the project finished – is this now learnt behaviour ?

One of the initial starting points that was stressed to all teams at kick-off was the fact that for this study the key driver was quality. For this study, this was defined as providing full coverage of the product under test and to find the largest number of valid defects as possible with the focus to be on the higher severity defects. All teams were given the same definition and understanding of the ratings for severity.

Each team was given as long as they thought that they would need in order to achieve the above stated goals for quality. This did not apply to the Bug Hunt teams, as part of the process for this is that they are only given two days in which to complete this service. It should be noted, however, that both Bug Hunt teams did overrun by one day each and this has been factored into the metrics.

Interestingly, we did see some unenforced team competition between the two Agile teams. Team 1 completed their project before Team 5 had started as we were waiting on staff to fit the criteria we had set and for the tool to be configured as we required. Team 5 self imposed the same timebox as Team 1, even though they were told repeatedly that time was not the driver but quality. As Team 5 had an increased learning curve to use the unfamiliar tool they did not complete as much testing and therefore their coverage and number of defects was significantly lower. This highlights that the Agile teams need to be given time to learn a new tool as well as understand the process. It also raises an interesting question as to whether testers now have learnt the behaviour to finish as quickly as possible and that quality is not the key driver – is this in fact learned behaviour due to the continual drive to squeeze testing? This is another area that I would like to do some more investigation on.

Tool Set

Bug Hunt

Both Bug Hunt teams used a Planit defined process for this service. They used template session sheets to record the testing and template excel sheets for the defects. This was using a light weight process as the methodology.

JIRA

Planit has configured JIRA, with its own test process workflow and pages, to reflect the assets that need to be utilised depending on the methodology undertaken. There were separate versions that were setup for each of the Agile and the Traditional teams. These had both been used for previous projects and were therefore proven. Each team that used JIRA had one member of the team that had helped with updating the standard configurations, so were very familiar with the workflows. The rest of the team were given a 2 hour training session on the templates as they were already familiar with how to use the basic JIRA functions.

The Agile version constituted user stories, tasks, session sheets, roadblocks and defects. It allowed the team to have a backlog and move stories into iterations in which they would be actioned. The traditional version aligned to the Waterfall methodology and allowed the creation of requirements, test conditions, test cases, risks, issues and defects. The tool allowed the linking of each asset back to the requirements.

HP Quality Centre Accelerator

Although all of the team had used Quality Centre v10 extensively on previous projects, they found using the Accelerator confusing. They considered it a large overhead entering data that they considered un-necessary and the automatic creation of, for example, folders which the tool does in the background far too heavy weight for an Agile project. They had all been using JIRA and considered this a more appropriate tool for light weight methodologies. It should be noted, However, that they did comment that had the project been longer and more complex there may have been some justification for some of the overheads.

It was evident that the amount of training that we should have put in place for this tool should have been greater and this would have benefitted the team. They did have the same 2 hour training session as the other teams on their tools. This would then have impacted on their iteration zero timebox. Lessons have been learnt from this and a training manual produced and updated with the key questions and answers to help this going forward.

Conclusion

Whilst there were some clear findings, lots more questions have been raised. The plan is to extend the study to focus on some of the anomalies found to date and also provide further data to confirm our conclusions.

The Agile team found the most defects, including the largest number of high severity defects. Their focus was much more on the user experience, whilst the traditional teams found more at the database level.

The bug hunt teams certainly represent value for money with much shorter timeframes and therefore cost. They give a window into the quality of the project as a whole and can be used to then target further testing from their results.

People are the most important. The most overwhelming discovery from this study is that the quality of the staff seemed to make one of the biggest contributions to the success of the project rather than the methodology used. If you have experienced professional specialists and they can collaborate continuously in a team environment, they will find the most high severity defects. It is those that are not good communicators and do not interact well who will contribute to making the team less effective.

The team needs to take care when selecting tools to make sure they are complimentary to the methodology and sufficient training has been put in place. For the study each team was not given a choice as to the tool they thought would be best to support them, but were given the tool. Each team was given the same length of training for the tool. We found that some tools were better suited to the project we were working in. In this case light weight tools were better. Also the amount of time required for training should be carefully considered to ensure that this is factored into the project timescales.

Depending on the testers involved, there may need to be some initial support of a new process in the case of Agile. This needs to be considered to ensure that the process and practices are embedded into the team or the team has some early coaching/mentoring for the first few iterations.

Surprisingly, seating arrangements could play a very important role in success. For this study the team seating seemed to have had a marked impact on the way the team worked together. Where the tables were grouped and the team all sat around the same table facing each other, their conversations increased which resulted in more significant defects being found.

Further data is needed to collaborate suggestions and over the coming months the project will be run again a number of times setting the environment into selected states to try and prove or disprove hypotheses.

If you have any similar data, please send it to add to this study – lhoward@planit.net.au

> biography

Leanne Howard
is an Account Director with Planit. Over the past twenty years Leanne has worked in the IT Software Testing industry across multiple projects including those using Agile. She is a specialist in the implementation of practical testing methods with a strong focus on client satisfaction. She enjoys sharing her knowledge through mentoring and coaching others to promote continuous learning, and is active in the testing profession.

Break free!

freedom learning
beyond the classroom

– a unique cloud-based learning solution – simple monthly licensing

- Save 90% on total cost of learning
- On-demand, 24×7 access to global testing training content and resources
- Replicate classroom benefits
- Blend Self Study with Virtual Classrooms and Real Classrooms
- Virtualization with real tutors, real time and real benefits
- Increase certification pass rates
- Value-based licensing – low cost of engagement and implementation
- Scalable, global, secure, private & proven

Visit www.learntesting.com or email enquiry@learntesting.com to find out more about Freedom Learning

Please quote TEXMAG for an additional **10% discoun**t off all Learntesting licenses for Testing Experience Magazine readers

te testing experience

www.learntesting.com

learntesting

Does Model-based Testing contribute to Business IT alignment?

by Colin Lek

I used model-based testing in a mixed context. But what were the contexts? After a review of literature on the Internet, I often identified a context where Model-Based Testing (MBT) appeared to be a feasible approach to control the software quality and reducing the costs related to the testing process, because test cases can be generated from the software artifacts produced throughout the software development process [1].

I found another context at a company. When I began one of my projects, I faced a problem: How can I apply a MBT approach when there is no formal model, e.g. finite state machines, UML diagrams, description of the software or system behavior?

This project, conducted at a financial company in The Netherlands a few years ago, quickly answered the question of what it takes to start MBT from scratch?

Our system analyst, a member of the requirements team, and me as the Test Analyst analyzed a mainframe system and traced its creation back to the 80's. This system had become an important source system in the chain of systems within and outside the company. The system analyst used pseudo code to document the system and the changes into a "functional design" to be used as basis for the development and test teams. Pseudo code is a compact and informal high-level description of the operating principle of a computer program.

For me it was hard to review this test base and to create test cases based on this functional design just by reading algorithms.

On the basis of the algorithms I first created a process flow model to get insight into the system under test. The modeling was done in MS Visio. The "if-then" and "if-then-else" statements from the pseudo code were transformed into a model.

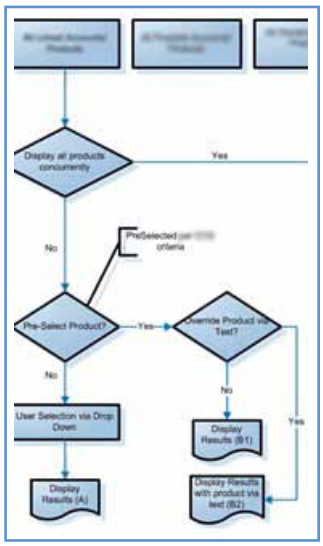


Figure 1: The modeling in MSVisio

To continue the MBT, we used the COVER test tool [2], designed by a Dutch testing company. The objective of this test tool is to make testing better, faster and cheaper.

This automation could support our test activities, such as test specification and execution. The test tool supports the test technique of Process Cycle Test (PCT), which could be perfectly used for test specification of the statements.

Secondly, derived from the process flow diagrams, the test tool automatically created test path combinations to execute the tests. It generated test cases directly from MSVisio.

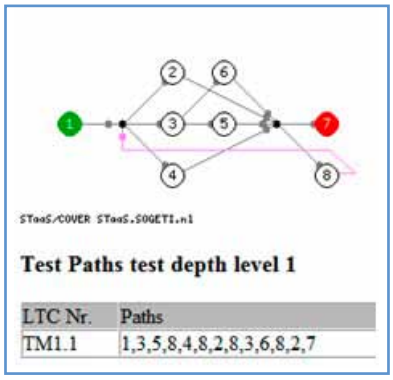


Figure 2: Test specifications generated by COVER

Getting back to other stakeholders in the project, for example business (owners) like information management and functional application management, they need the same functional design for the formal approval and acceptance of the IT project deliverable, as created by the system analyst. This project deliverable was a bridge too far, because they were unaware of this kind of system requirement. They were not able to interpret the expected functionality and discuss it with IT.

This meant that Business and IT had to communicate more effectively to achieve a true fit between the business' needs and technology's support (business-IT alignment).

In response to this, the test coordinator in a project meeting said, "The test analyst can play a role!" So I joined the project team not

just to test the system, but to also share my process flow model with the business. There was an interesting iterative process going on, which consisted of the follow activities: I designed process flows which were then reviewed by the system analyst of the system. The next step was eventually redesign the functional design. The organization and running of workshop meetings with business and IT to review and discuss our project deliverables continued for a few weeks. Business and IT people were drawing lines on the hard copy of the process flow models. These flow charts were easy-to-understand diagrams showing how steps in the system fit together. The flow diagram, originally an input for the test tool of the MBT approach, became a useful tool for communicating how the system works, and for clearly documenting information flows. It also helped to highlight where the system can be improved. It became clear that this was a good way of accepting the functional design. More important, the risk for the business in discovering what they wanted, and whether the design matched these needs or not, was reduced.

Now, years later, I can see this MBT approach, apart from reducing costs, also benefitted the (testing) process in other ways, such as reviewing the functional design and reverse engineering of a legacy system. By transforming the algorithms into a model we could trace inconsistencies and incompleteness. An unexpected discovery is that the testers can contribute to the alignment between business and IT as an ongoing process. This requires specific requirements capabilities (design of process flows) and involves holding workshops over a period of time.

Figure 3: Business IT Alignment improved?

We have added value to the project by combining the knowledge of both worlds and bringing them together.

We spend a little extra energy trying to align people; the momentum was during the test specification phase in the project where we killed two birds with one stone. We reduced the testing costs through the MBT approach, and bridged a gap between business and IT.

References

[1] A.C. Dias Neto, G.H. Travassos, R. Subramanyan, M. Vieira (2007), "Characterization of Model-based Software Testing Approaches", Technical Report ES-713/07, PESC-COPPE/UFRJ. Available at http://www.cos.ufrj.br/uploadfiles/1188491168.pdf.

[2] TSTR, (2012). 'Cover Sogeti Test Tool' http://tstr.nl/cover/index.php?language=en

> biography

Colin Lek
is Test Consultant and partner at The Future Group. He received his Bachelor of Business Administration and a Masters in research into Business & IT Alignment from the HU University of Applied Science Utrecht in The Netherlands. He has eight years of experience in the financial, electronics, production and public sectors. He has held several positions in IT and testing, including Test Coordinator.
e-mail: colin.lek@the-future-group.com
website: www.the-future-group.com

An introduction to model-based testing for the automotive industry

by Peter Farrell-Vinay

More complex and inter-connected automotive control and monitoring systems are factors leading to the increasing adoption of model-based testing. Automakers are undertaking model-based testing to reduce development costs and the time taken for new vehicles to come to market by validating complex aspects of the vehicle design in advance of manufacturing. Model-based testing also provides a means of identifying and modeling critical system functions. For example, timing issues in powertrain and chassis assemblies and safety-critical problems in vehicle acceleration control systems can be discovered before a single spanner turns, again reducing design and integration effort and costs.

Shared insight

Although many of today's automotive systems stem from railway or aircraft technologies, such as anti-lock braking systems and engine control, the requirements of these systems in vehicles differ from those on trains or airplanes as, in automobiles, all electrical, mechanical and software systems rely on each other. Each vehicle is autonomous, with no external guidance or information systems, so model-based testing allows the auto industry's software, electrical and mechanical engineers to find a common ground in the design phase, by working together to develop high-level models. These high-level models are then used for simulation purposes, detecting problems in the very early stages of the development process and offering a shared insight that the alternative of writing out a detailed design simply doesn't provide.

The MBT process

There are many MBT process models. Here is a simple view:

Models are very tool-dependant and some tools allow for a series of model decompositions in which a simple model becomes

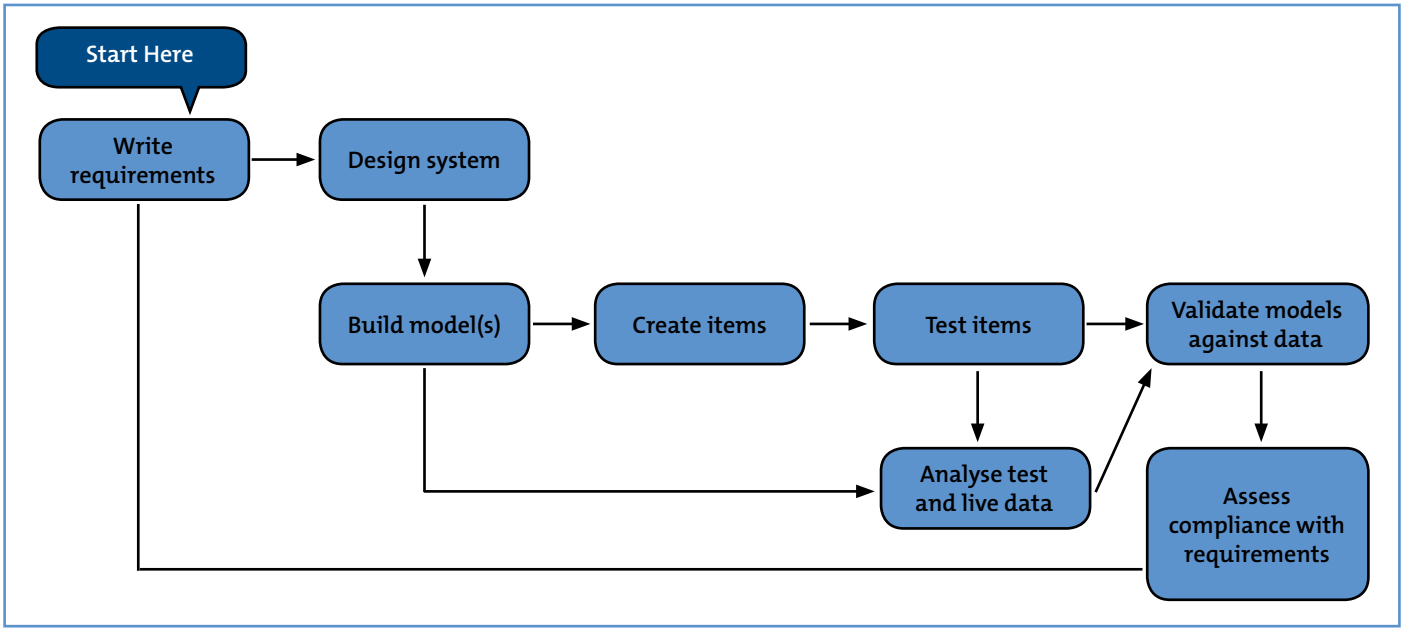


Figure 1 - An idealized model-based development and test process

increasingly complex and representative. These decompositions (shown above as "items") allow for different test types ranging from logical models through to the code. In a well-structured environment, once the models have been validated, they can be substituted by real hardware or a hardware simulator.

Pre-design requirements

High-level models are treated as a critical part of the pre-design requirements as they help identify how the design should work and allow the creation of more advanced tests, ensuring that all the integrated systems behave as required and expected.

Testers begin with as simple a description as possible of the functions to be modeled and, as the model grows, they add the detail about all the unmodeled interfaces to this description. Each change to the model is recorded as an individual decision so that, at the end, the models being used to test the vehicles have functions that can be exhibited and traced to specific / different decisions. Each test of those functions can also then be traced back to the part of the model or interface that it covers.

The different models within model-based testing give testers the ability to 'run' a host of scenarios which validate any complex and/or safety-related systems and their competence in preventing dangerous states from occurring. In addition, a model can be used for this without the testers ever having to build a physical system to put it into such states and determine how the vehicle would behave. Of course, not all vehicles are created equal; heavy goods vehicles typically have more axles, wheels and brakes than a car, but otherwise the principles of model-based testing are the same for all vehicles, whether they be buses, cars or vans.

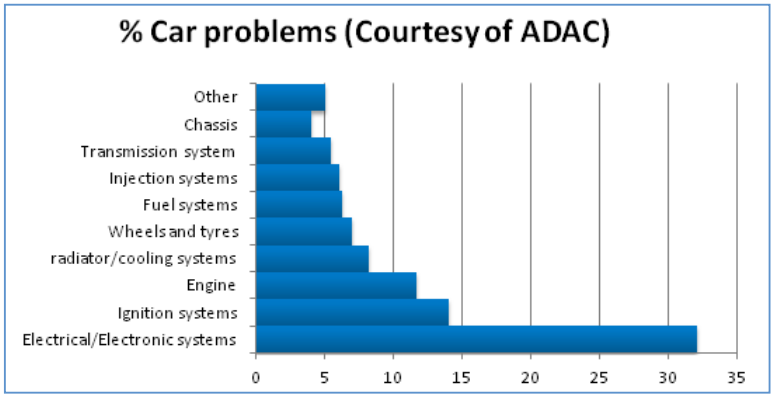
Model-based testing is modular and offers an inexpensive way of carrying out rigorous requirements and design testing at each level of integration. A good example of early-use model-based testing in vehicles is the identification of issues such as the potential locking of a vehicle's acceleration long before it may have been found by physical or software tests later in the manufacturing process.

Embedded software

Product development is increasingly prone to complexity as the number of products built around embedded software is growing rapidly, together with the embedded software's functional capabilities. Embedded software thereby frequently displaces functions previously popular in hardware such as digital fly-by-wire flight control systems, which were superseded by mechanical control systems. Software also increasingly enables new functions such as intelligent cruise control, driver assistance and collision detection systems in high-end vehicles, so that today's average car contains around seventy computer chips and over 500,000 lines of code.

Maintaining brand image

While high code volume itself is not a problem, the increasingly complex interactions between components and subsystems are. Software glitches and bugs that emerge only after vehicles have entered production mean a rise in warranty and testing costs as well as sometimes irreparable damage to established and new automotive brands. Malfunctions within the automotive industry, particularly electronic components, account for as many failures as all other automotive systems put together; so, without model-based testing to identify safety-critical problems as early as possible in the manufacturing process, costly remedial work and damage to the manufacturer's brand image – and indeed the vehicles' sales – is all but guaranteed.



Conclusion – are we there yet?

Being "there" means how powerful your models need to be:

- Modeling tools based on **executable UML** (build the model in XML, generate code, generate tests, execute the tests, correct the model).
- Modeling tools creating representation of closely-tied hardware and software functions, such as Prover, which can generate proven models of limited systems and generate outputs (like railway signaling routes) or which show problems in existing systems through detailed modeling.
- Models of requirements which can **automatically generate tests**, such as AutoFocus.

> biography

Peter Farrell-Vinay
is Managing Consultant at SQS, the world's leading specialist for software quality. He brings over 35 years of software testing and quality assurance experience across a wide range of enterprise IT and embedded software projects.
Peter has strong language skills and has worked in a variety of geographies, being part of and leading teams ranging from small developments to some of the largest in the world. With a proven ability to review and test software, and as the author of "Manage Software Testing" and various papers, Peter's past work experience includes being a test manager and mentor for many leading companies including CSC, Reuters, Metastorm, Categoric, Logica, Unisys, British Telecom and Alstom.
Peter also has a proven track record of achievements and a strong exposure to safety critical aspects of quality and quality process improvement.

Model-Based Testing – Practical applications and use in the communications industry

by Kalilur Rahman

Testing as a discipline has made tremendous progress over the last two decades, with the advent of innovations across the IT industry and with regard to the maturity in delivery approaches, tools, technologies and proven measures. Testing as a practice has also been innovated continuously. One of the cutting-edge, proven mechanisms is Model-Based Testing (MBT).

If followed and implemented correctly, MBT offers multiple advantages for testing teams, although there may be some minor disadvantages and overheads. Some of the key benefits of MBT are:

- Each model used for MBT can be aligned with the business process (and can be an extension of a domain/business model or framework, such as eTOM, ARTS, etc.)
- Requirements can be represented visually in the form of a model (some of the prominent ones being state diagram/finite state machine/graph)
  - We could use state model for representing complex business processes in the form of a graph or a "modularized state model"
  - Graphs or state models can be assigned weightages or values based on business priority/criticality and complexity
  - Requirements can be visualized in an end-to-end manner
  - The models can be integrated into a common infrastructure/framework with industry-accepted tools
- MBT will also allow for visual static verification testing in order to identify requirement gaps using a "shift left" approach, and also to reduce the cost of quality for the SDLC
- MBT paves the way for automated test case generation, thus reducing cost of testing/test preparation
- Allows for ease of test case/automation maintenance
- As visual modelling is possible, MBT can be merged with pair-wise testing/risk-based testing to pick optimal/business centric test scenarios relevant for testing

The key disadvantages of MBT are:

- Steep learning curve and the challenge related to understanding
  - Industrialized/common understanding between business analyst -> designers -> testers and the personnel

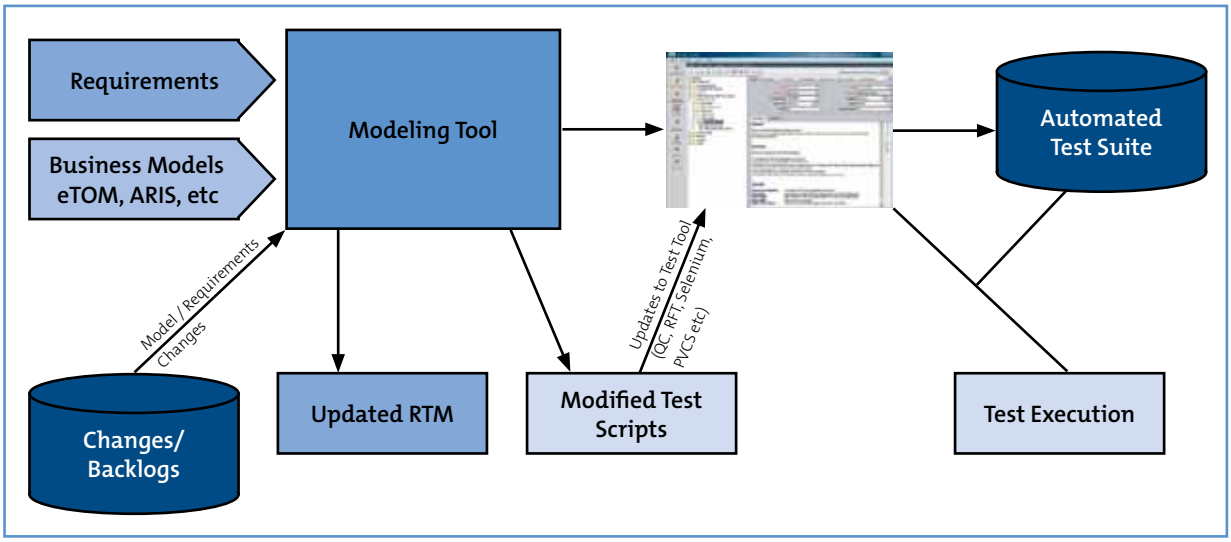


Figure 1 Model-Based Testing - Approach

creating test cases using automated tools

- Lack of fit-for-all testing phases/types/conditions/purposes/requirements due to MBT's unique nature
- Challenging implementation inertia due to technical complexity, need for mathematical knowledge is in some cases required

There are many success stories related to model-based testing – one of them being the reduction in testing costs/test automation maintenance costs for web-sites/workflow management to regression testing of communication protocols for routers, etc.

Key Advantages of MBT

- Reduction of test script maintenance effort – very useful for applications/solutions involving multiple state model/transition modes
- Reduction in test prep/review and execution effort – due to repeatability and consistency of scripts generated using "proven" models

As a part of model-based testing, the test scripts are generated in automated fashion using tools. This approach by MBT allows for more automated test scenarios to be generated and executed. In a normal testing life cycle, as the testing complexity grows due to incremental feature additions, the test requirements and test script maintenance become effort intensive. With MBT, this challenge is handled smoothly by the automated script development. The testing team can handle iterative/agile feature enhancements by matching with incremental complexity for the testing model (using graphs, finite state automaton, or state transition charts for example). This enables testing to match the pace of development, especially in case of feature/model-based environments (e.g., for a product for telecom protocol or telecom mediation).

This idea is hypothesized in the diagram (figure 2).

However, when we apply MBT in end-to-end testing phases (such as systems integration testing, or in user acceptance testing), MBT becomes a challenge to implement and manage.

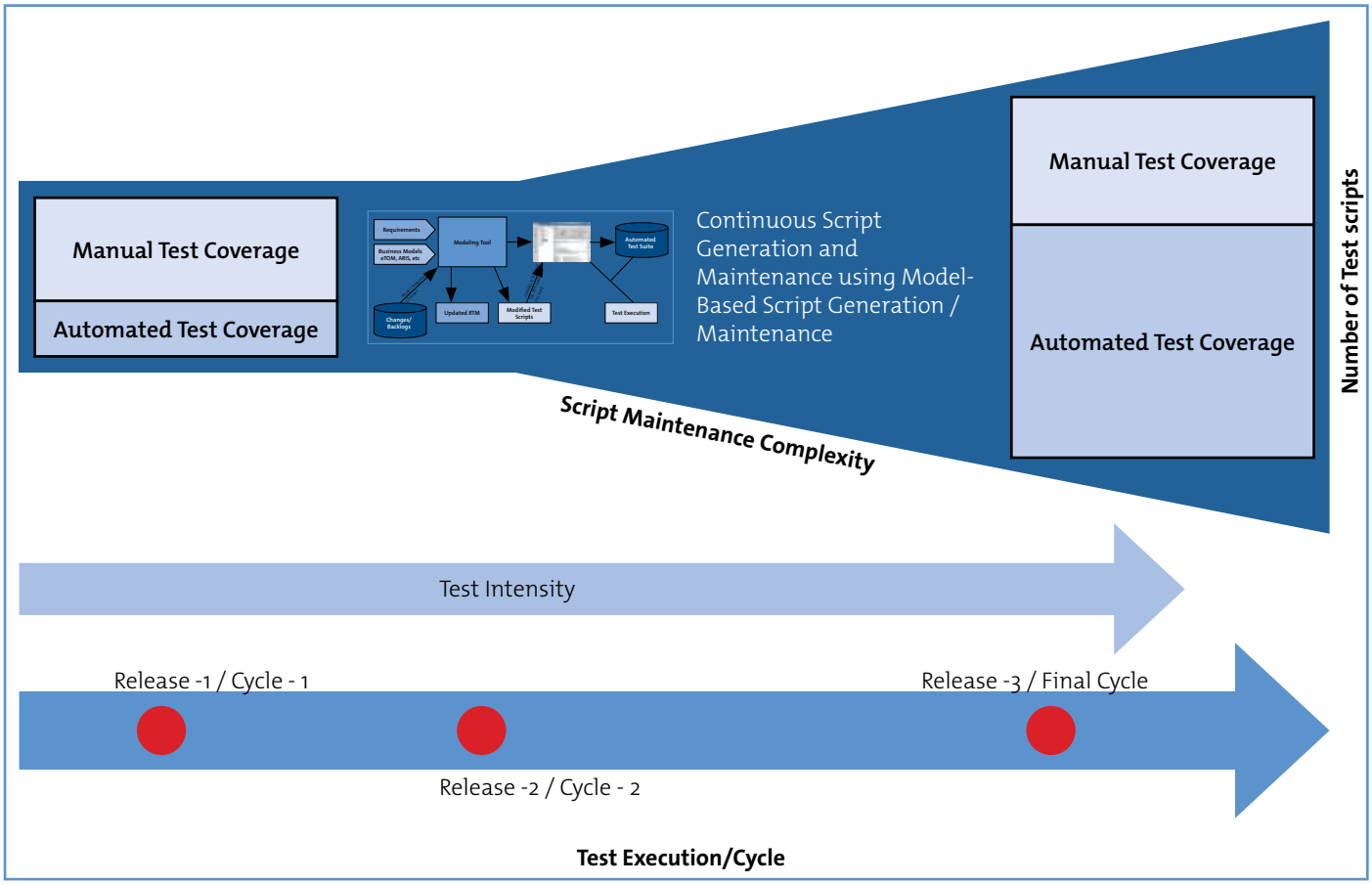


Figure 2 Usefulness of MBT in complex projects

One way to implement MBT in these phases would be to align the test requirements using a state transition model, the state of an actor/user, and base the scenarios according to the transition of the state of the actor. One example for a telecom customer (in a hypothetical scenario) could be as shown in Figure 3.

If we use automated scenario generators/technologies, such as Spec#, conformiq, AsML, SpecTest, Lisa Pathfinder, QC/BPT, RFT (to name just a few) or other commercially available tools, the state models of an actor (customer) could be represented and test scenarios could be generated automatically.

One way model-based transition could help in end-to-end system test phases, can be understood by an example: In telecom mediation for Call/Event/IP detail records and their use for billing, the CDRs go through a state transition. If you take a commercial CDR mediation system, the system could have more that 10-15 states depending on the type of implementation.

Let's assume we have a challenge of testing a mediation platform roll-out with incremental features/states – for example states 1,2,4,5,14,3 for iteration-1, and states 1, 2, 4,5,7,9, 12, 14, 3 for iteration-2, and the entire lot for iteration-3. In this case, the use of

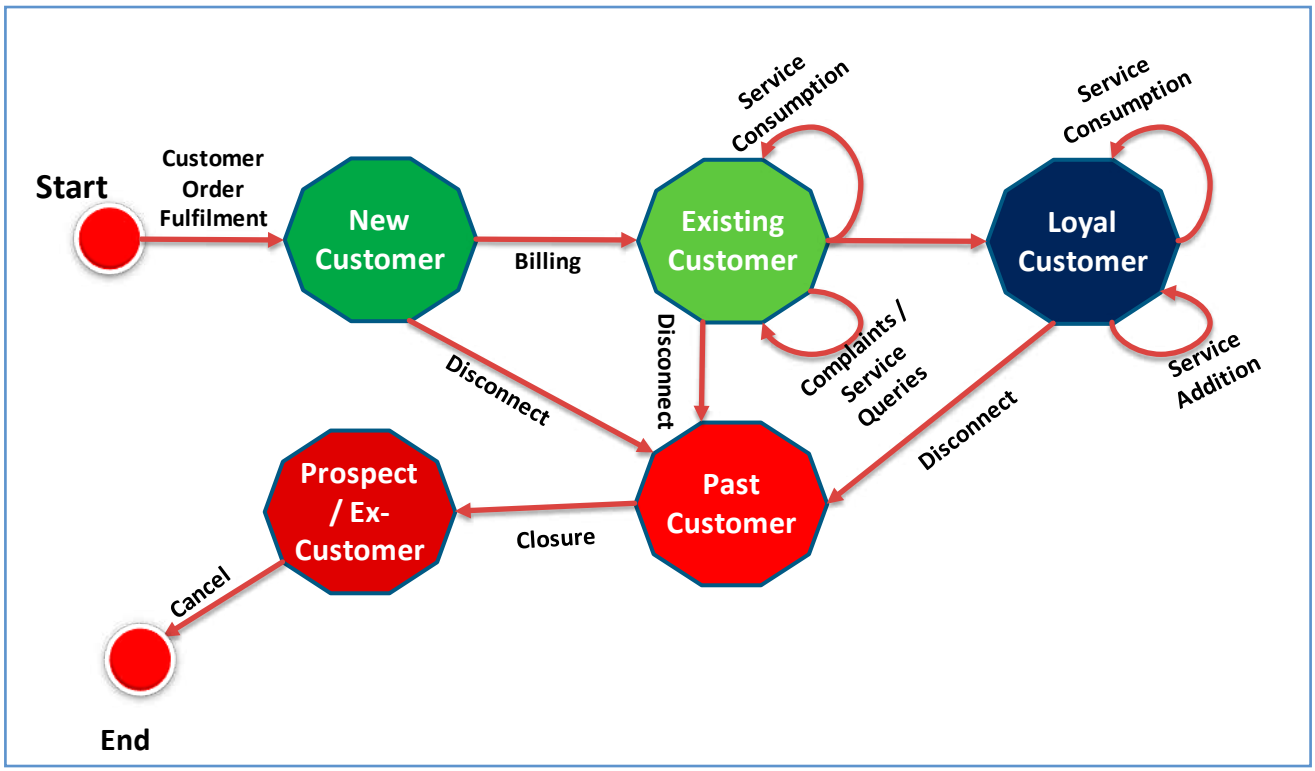


Figure 3 Customer State Transition Model for a Communications Industry Scenario

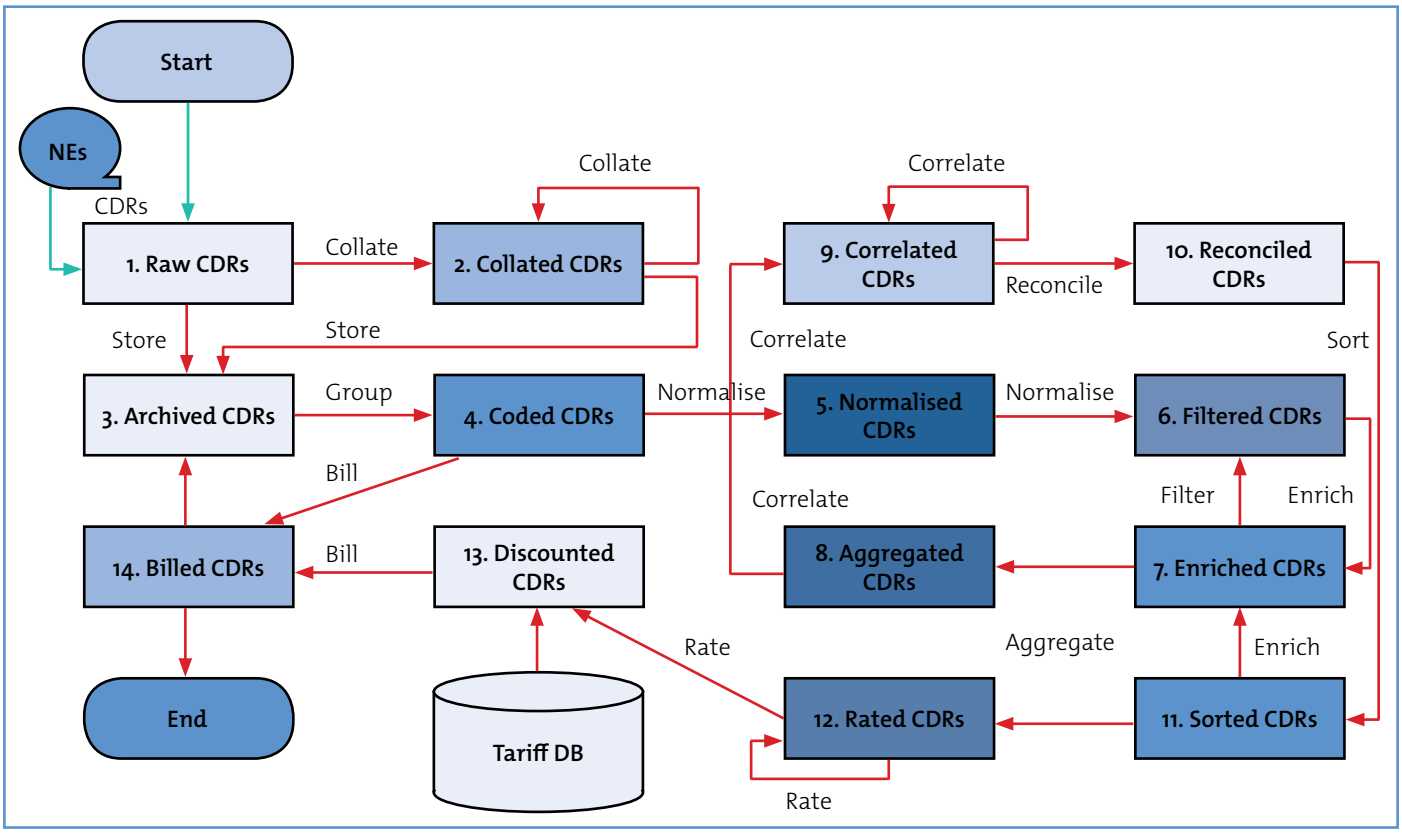


Figure 4 State Transition Diagram for CDR Collation

an automated tool generator (as specified above) or the use of a pair-wise tool such as Hexawise, will make script generation a lot easier and reduce test preparation effort. These state transition functions can be developed incrementally (such as collate, correlate, normalize, rate, store, filter, enrich, sort, etc.).

Similarly, if we want to map a state transition model in a tabular fashion, this could be represented in the table below which shows a a hypothetical example of a customer order management application in the communications industry (e.g. for a CRM system). If we implemented a state-model for a communications customer order state transition, one of the permutations may look like the following table:

Communications Customer Order Management – State Transitions															
	Initiate Contact	Save Interaction	Feasibility Check	Credit Check	Enter Order	Submitted	In-Process	Error	Completed	Cancelled	Amended	Fulfilment Completed	Billing Initiated	Change	Disconnect
Initiate Contact	x	1	2	4	5	6	7	8	9	3	12	10	11	13	x
Save Interaction	x	x	x	x	x	x	x	x	x	15	x	x	x	x	x
Feasibility Check	x	x	x	16	17	18	19	20	21	27	25	22	23	24	x
Credit Check	x	x	x	x	27	28	29	31	30	32	33	34	35	36	x
Enter Order	x	x	x	x	x	38	39	40	41	42	43	44	45	46	x
Submitted	x	x	x	x	x	x	48	50	49	51	52	53	54	55	x
In-Process	x	x	x	x	x	x	x	58	57	59	60	61	62	63	x
Error	x	x	x	x	x	x	x	x	69	73	67	68	70	71	72
Completed	x	x	x	x	x	x	x	x	x	74	77	78	79	76	x
Cancelled	1	x	2	3	4	x	x	x	x	x	80	x	x	81	x
Amended	x	x	x	x	x	x	82	83	84	85	x	x	x	x	x
Fulfilment Completed	x	x	x	x	x	x	x	x	x	x	x	x	86	85	88
Billing Initiated	x	x	x	x	x	x	x	x	x	x	90	x	x	89	91
Change	1	13	x	x	x	x	x	x	x	93	x	x	x	x	92
Disconnect	1	2	x	x	x	x	x	x	x	x	x	x	x	x	x

Using a state-modeling language, it is feasible to represent the state models for customer orders based on the requirements, and plan design, development and testing based on priority – by focusing on the incremental features. This could help in incremental, iterative feature development and deployment for large transformation programmes.

With standard industry models, such as eTOM for telecom and similar ones for other industries, MBT can be an effective tool - if used wisely and in conjunction with complementary tools, such as Hexawise etc.

As a conclusion – MBT is here to stay. MBT may be a little complex, challenging to understand and may not applicable to all types of testing. However, it has been proven in a multitude of ways – in devices, protocols validation, unit testing of GUI applications and product development. There are major opportunities in the way MBT could be used for end-to-end testing, automation for reduction of testing costs, and for shift-left verification and validation.

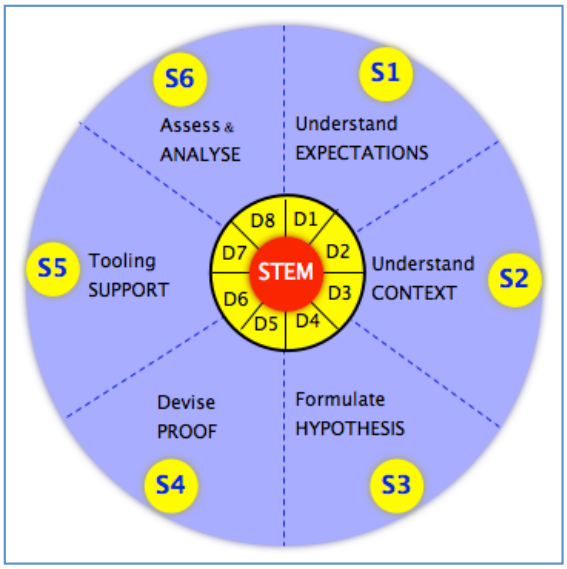
> biography

Kalilur Rahman
is a Senior Manager working for Accenture. He has 15+ years of experience in IT – primarily in the communications industry. Before Accenture he worked with Cognizant and TCS. He Following his graduation in Computer Science & Engineering, he has played roles across all SDLC phases. He was a key founding member of a large end-to-end testing deal for a leading communications company and is extending the process improvements to another transformation engagement. He is passionate about writing and sharing his knowledge. He is an active member in the TeleManagement forum and a lot of internal initiatives. He can be reached at rahman.kalilur (AT) gmail.com

Hypothesis Based Testing – A personal, scientific methodology to defect detection

by T. Ashok

Testing is seen as a key driver in producing clean software. A variety of approaches have been formulated to ensure good testing to deliver quality software. These various approaches can be process-centric, domain-centered, tool-driven, context-driven and creativity-based. All these approaches typically focus on the activities related to the test life cycle.



Hypothesis Based Testing (HBT) is a goal-centered approach that commences by identifying cleanliness criteria and the corresponding potential defects and then performing a set of activities to ensure purposeful testing, which is effective and efficient. The central theme is to construct a hypothesis of likely potential defect types and then scientifically prove/disprove them.

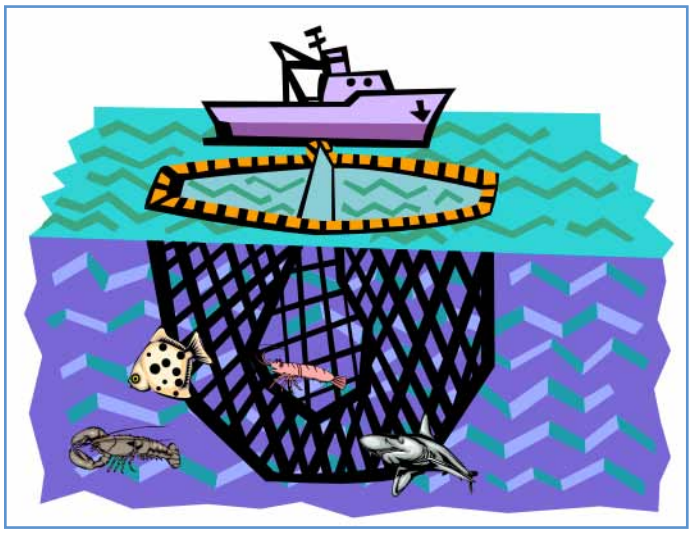
How Is HBT Different From The Typical Test Methodologies In Vogue?

Typical in vogue test methodologies have relied on strength of the process and the capability of the individual to ensure high quality software within the given cost and time constraints. They lack the scientific rigor to enable full cost optimisation and more often rely on automation as the means to driving down cost and cycle time. For example, often they do not provide a strong basis for assessing the quality of test cases, and therefore the effectiveness and efficiency of their defect finding potential.

HBT on the other hand, enables you to set a clear goal for cleanliness, derive potential types of defect and then devise a "good net" to ensure that these are caught as soon as they get injected. It is intensely goal-oriented and provides you with a clear set of milestones, allowing you to manage the process quickly and effectively.

Hypothesis Based Testing – A Fishing Analogy

Consider a fisherman who fishes in a lake and wants to improve his earnings. How can this be accomplished?



1. Catch more fish.
2. Catch those fish that can fetch a higher rate.

To accomplish the objective of earnings improvement, it is good to start with (2) and then (1). This can be done via:

3. Identifying what kinds of fish you want to catch
4. Understanding where they frequent/congregate/live
5. Identifying net size and the net's hole size
6. Understanding technology that could help in catching fish

7. Figuring out how much of the lake you want to cover
8. Figuring out what kind of boat you want.

Steps 1-2 focus on (2) while 3-6 focus on (1).

The key points here are setting the goal of what you want to catch and then figuring out means to catch them. The fish are analogous to the potential defects in the software. To detect these defects scientifically, we need to hypothesise the defect types, formulate the types of tests to be done, identify the earliest stage of detection, and then identify the techniques/tooling needed to perform this.

HBT – Six Stage Evaluation Methodology

The act of validation in HBT is a six stage process. It commences with two stages focused on a scientific approach to understanding the customer's expectations and the context of the software. One of the key outcomes of the first two stages is "Cleanliness Criteria" that gives a clear understanding of the expectation of quality. In the third stage, the Cleanliness Criteria and the information acquired in the first two stages are used to hypothesise potential types of defects that are probable in the software. The fourth stage consists of devising a proof to scientifically ensure that the hypothesised defects can indeed be detected cost-efficiently. The fifth stage focuses on building the tooling support needed to execute the proof. The last stage is about executing the proof and assessing whether the software does indeed meet the Cleanliness Criteria.

Stage #1	Understand EXPECTATIONS
Identify the marketplace for the target system/software, the business need that the system/software is expected to fulfill, various end users who will use the software/system, and gain an understanding of the technologies used and the deployment environment. In our fishing analogy, this is about understanding the biodiversity of the lake, area/size of the lake, type of water in the lake, and the kinds of fishermen.	

Stage #2	Understand CONTEXT
Understand the various business use case (or flows) for each end user, the corresponding technical features (for each requirement), the attributes to be met for each feature/requirement, the usage profile, and the prioritization of business value. Using the fishing analogy, this is about understanding the types of fishnet/bait used, and challenges in fishing like areas/times fishing is allowed.	

Stage#3	Formulate HYPOTHESIS
Hypothesise the potential defects (PD) in the software based on the technology, deployment environment, usage, implementation and the domain. The outcome is a list of potential defects that are aggregated by potential defect types (PDT). This helps us to clearly understand what could cause dirtiness in the software and map the PDT & PD to the technical features and business requirements. In the fishing analogy, this is about understanding the kinds of fish/aquatic animals that breed in the lake.	

Stage #4	Devise PROOF
Having identified the potential defects and defect types, the objective is to identify the earliest point of detection in the software development lifecycle. The classic methods of doing this have been to define testing at three levels – Unit and System Integration testing in the development environment and Acceptance testing in the customer environment. In HBT, there are NINE quality levels, with each level focusing on specific potential defect types. This allows the test cases at each level to be sharply focused and enables them to be very effective. It also allows for easier scripting, with scripts being shorter and simpler. The test cases at each level are designed using a pragmatic behavior model-based approach, and the test cases are traced to the potential defect types that they can uncover leading to a the new concept of "fault traceability" that allows an objective assessment of test coverage. Finally, design the measures (metrics) to ensure test adequacy, test progress and health of the software/system. In the fishing analogy, this shows where (what areas) to fish, types of bait to use, types of nets to use, what kinds of fish to catch initially and subsequently (e.g. catch small fish on the surface near the lakeshop before deep-lake fishing using a boat), tools needed for fishing (sonar, boat) and then devising a cost effective net that is broad enough and deep enough to ensure that those fish can be caught. Finally, it is designing measures to ensure that the fishing strategy is indeed working (number of fish, types of fish caught, rate of catching, etc.)	

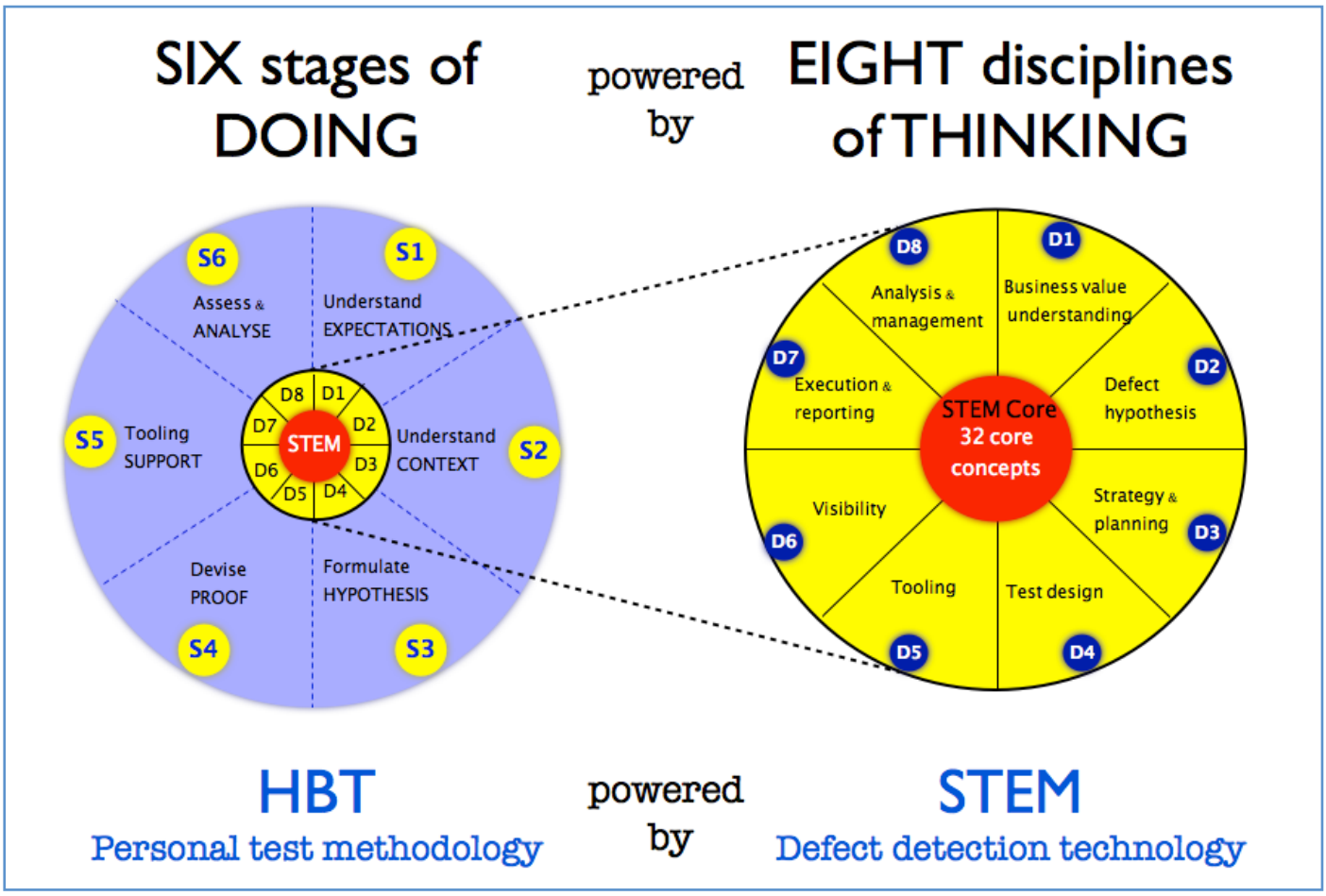
Stage#5	Tooling SUPPORT
Identify any tooling requirements needed for executing the proof (i.e. execution of scenarios/cases). Identify specialised test bench that may need to be developed to execute the test scenarios. Identify potential scenarios/cases that need to be automated, and devise the strategy for functional and non-functional test automation. In the fishing analogy, this is about identifying suitable technology (like sonar or a faster boat) to ensure that the fishes can be effectively and efficiently caught.	

Stage #6	Assess & ANALYSE
Now execute the identified scenarios/cases for each cycle, log defects and assess the cleanliness criteria. Collect the various measures identified and analyse the test progress, health of the system/software and the business risk to meeting the delivery deadlines. Using the fishing analogy, this is about performing the act of fishing and then analysing the fish caught to understand whether we are catching the right types of fish, the rate of fish catching and performing appropriate course corrections.	

STEM™ – The Method That Powers HBT

HBT is a scientific personal methodology for testing software/systems, whilst STEM is a method that enables the various activities in each stage of HBT to be performed in a scientific manner. STEM consists of eight personal disciplines, where each personal discipline focuses on a key activity in HBT. Each discipline is a collection of how-tos that in turn rely on THIRTY TWO scientific concepts. These can be a technique or a principle or a guideline.

At the heart of STEM are thirty two scientific concepts that ensure that each discipline is implemented well.



STEM Discipline	Objective	Used in HBT Stage
Business value understanding	Understand application/product, users, usage, focus areas, intended business value	Stages 1, 2
Defect hypothesis	To identify the potential types of defect in a logical scientific manner	Stage 3
Strategy & planning	Identify the quality levels, test types, test techniques, execution mode, estimated effort, scope test cycles.	Stage 4
Test design	Design test scenarios and cases	Stage 4
Tooling	Analyze tooling needs and design tooling architecture	Stage 5
Visibility	To identify metrics for progress, quality and risk	Stage 4
Execution & reporting	To enable effective execution of tests	Stage 5
Analysis & management	To analyse quality progress and risk post-execution	Stage 6

Results Of Applying HBT

HBT has been applied to various projects and executed on diverse technologies in a variety of domains across different phases of the product lifecycle. The people involved had a mix of zero to five years of testing experience. We observed that HBT can be plugged into any stage or situation of the project for a specific need, and one can quickly see the results and get desired benefits as required at that stage.

Our experience and analysis showed that there are varied benefits of using HBT, such as rapid reduction in ramp-up time, creation of assets for learning, consistent ways of delivering quality results even with less experienced team members, increased test coverage, scientific estimation of test effort, optimisation of regression test effort/time/cost, selection of right and minimal test cases for automation and hence reduction in its development effort/time/cost.

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

Examples

SAAS / Bidding Software	
Characteristics	Java, Apache, JBoss , Oracle 9i with cluster server on Linux 300KLOC+, 4 month duration.
HBT Application	Three people with no prior knowledge of application were able to rapidly understand the application and derive the test strategy in about 60 hours. 255 potential defect types were identified in 45 hours. The rapid understanding was possible by applying "Landscaping", a core concept based on mind-mapping. 3 people designed 10750 test cases from these, 7468 (69%) positive test cases and 3282 (31%) negative test cases. 9 people were involved in test execution, 12 builds were tested in 3 iterations and 4 cycles, 2500 defects logged. 500 defects were of high severity.
Results	No bugs were found in UAT. Rapid reduction of regression test cycles due to use of "Interaction Matrix" STEM core concept. The test method, structure & templates derived are being used as a reference model by the customer in other projects. We found critical defects that ensured that revenue targets were met due to the efficiency of the process.

Supply chain application in the maintenance phase	
Characteristics	A web service application with five major features that had frequent enhancements and bug fixes. Application built using Java, Apache. Project duration of four weeks. Application is in production and therefore is in the maintenance phase.
HBT Application	Application of "Landscaping" and "Interaction Matrix" core concepts deepened the understanding of the product and understanding of the impact of other services and usage on this service. Defect hypothesis yielded 118 potential defects in less than a day.
Results	Reduced ramp-up time for the induction of new testers - from 16 hours to 4 hours. A new member was productive from day #1 and could start testing enhancements and bug fixes on day #1. Listing of potential defects enabled addition of missing test cases from the existing test case set. A few defects were discovered in the production version of the software that related to workflow. These had previously gone unnoticed and were stated by the customer to be very important. In the fishing analogy, this means: we caught new types of fish that we were not looking for initially, devised different types of net and finally new fishermen were made productive quickly as they were focused on finding the right types of fish.

Project metrics tool	
Characteristics	A corporate tool that allows management and project managers across the world to track project status. Web based application (in its second major release). Two teams were asked to test the application, one employing HBT, the other not applying HBT. The numbers of people in each of the two teams were the same, and both teams were given the same duration for testing (one month).
HBT Application	All stages of HBT, except automation, were applied.
Results	The HBT team resulted in nearly 3x more test cases and 2x more defects. Not only were more defects found, the quality of defects were better from the HBT team. The HBT team found two residual defects in the (existing) application; one of them was serious enough to corrupt the entire data set. The team and the customer are able to rationally reason that support costs of the application would be much lower as application tested by the non-HBT team would have resulted in at least two patches to be delivered later. A clear demonstration of that the newer, goal-focused approach powered by a scientific method caught more and better revenue (yielding fish with no increase in cost of fishing).

Summary/Conclusions

HBT is a scientific personal methodology that enables a goal-focused approach to validating software. It allows for a scientific approach to testing software, enabling logical reasoning and questioning. In our applications of HBT we have seen results of increased coverage (3x), resulting in uncovering more useful defects (+50%) earlier in the testing lifecycle. This has in turn allowed us to reduce post-release escapes significantly (30-500%).

The scientific rigor of HBT has enabled us to rely less on experience for effective testing, helped us to scale the team faster, and lower the overall cost of testing. Over the last decade we have used HBT for testing applications across various domains, technologies and process models and therefore empirically believe that HBT is domain, technology and process neutral.

STEM is the trademark and HBT is the intellectual property of STAG Software Pvt Limited. STAG Software is an Indian based company that offers boutique testing solutions and are the offshore partner of Epicentre, a UK based software testing specialist. Information related to the HBT methodology and its application is freely available from both the STAG Software and Epicentre websites. If you would like to find out more about HBT, please contact Epicentre on hbt@epicentre.co.uk.

> biography

T. Ashok
Founder & CEO of STAG Software, MS in Computer Science (IIT Chicago), B.E. (Electronics and Communication) Anna University, CSQE (ASQC). Spearheads the operations of STAG, has 18+ years of working experience. Setup and managed global test group at VeriFone. Active in the test engineering community, has given numerous lectures in test forums and presented several tutorials/papers at international and national testing conferences. He is highly passionate about software test engineering and devotes time in his lab pursuing his interests in software patterns, test design, test automation, object technology, process engineering and measurement (including originating the Hypothesis Based Testing methodology). He was presented with the Thought Leadership award for his contribution to the testing community at the Test2008 international conference.

Two-Tier Implementation of MBT in Testing Service Oriented Architecture

by Prasad Ramanujam, Vinothraj Jagadeesan & Venkatesh Ramasamy

Background Details

In the wake of the double-dip depression, organizations are being forced to rethink their IT strategy. Unlike in the past when software budgets were generally unconstrained, organizations are now forced to squeeze their limited budgets to obtain maximum return on investments. The stiff competition in the market has forced organizations to pursue a more linear approach towards software development. Organizations are now on the lookout for delivery models that are faster than agile, and which eliminate redundancy from the entire process.

Nowadays organizations don't have the luxury of building a complex bespoke product from scratch to satisfy their business requirements. Therefore it comes as no surprise that they have started realizing that the solution to solving any complex problem is to break it down into simpler manageable chunks and solve each subset individually in order to solve the problem as a whole. So instead of building a bespoke product that satisfies all their business needs, organizations are now applying techniques like Model Driven Development (MDD) to segregate their business functions into separate entities. Each of these entities can then be satisfied with a specific Bespoke or Commercial Off The Shelf (COTS) product that exactly addresses the problem. This can be clearly understood with an example.

Sample Mdd Framework – Motor Insurance

Figure 1 below shows details of a high level list of entities that are involved with issuing a motor insurance policy. Typically, the system needs to validate the credit score of the user and also search the Motor Insurance Bureau (MIB) records for any past convictions. Once the details of the policyholder are validated, the system needs to validate the Vehicle Identification Number (VIN) to verify that the applicant has an insurable interest on the vehicle. Once both the applicant and the vehicle details have been validated, the system needs to calculate the applicable rates for the cover to be provided. Meanwhile the application details need to be stored in the organization's database and payment should

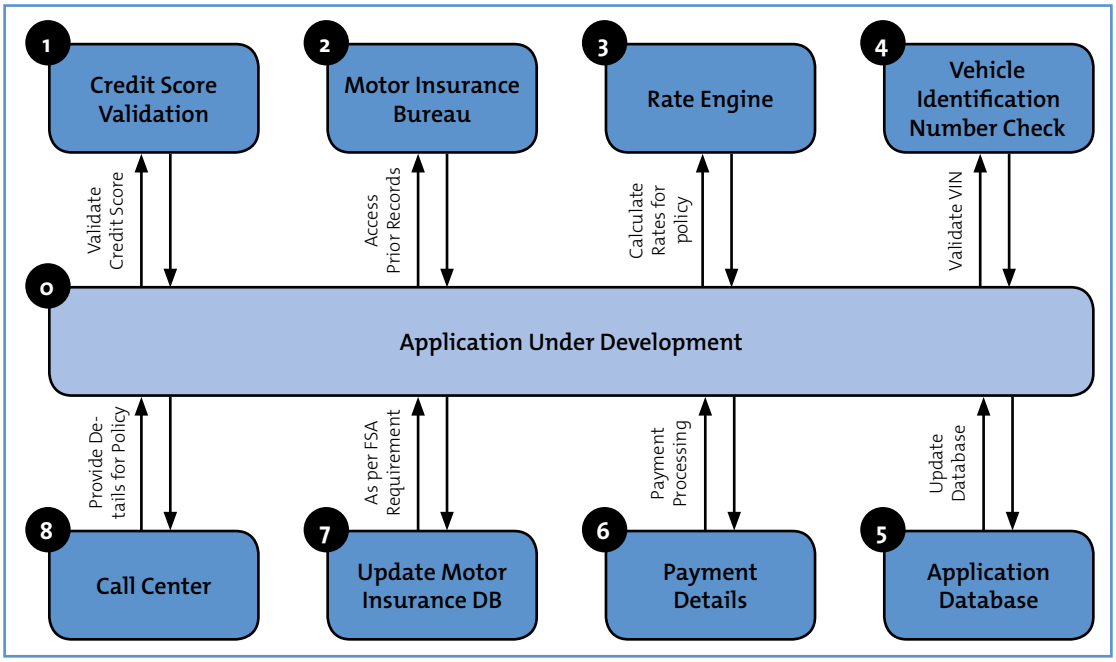


Figure 1. Sample Distributed MDD Models

be processed for the cover. Once the policy is issued, it needs to be registered in the Motor Insurance Database (MID) as per the rules of the Financial Services Authority (FSA). Finally, the details need to be made available for the organization's call center operations for future dealings. Therefore, typically, the operation of the system can be classified into eight independent entities, each of which has a specific input and produces designated output. In some cases, the output of one entity can be used as an input to another entity.

In the traditional approach, a Bespoke product would be created which contains all these entities as separate functionalities. In the newer MDD approach, however, each of the entities is satisfied by its own solution. For example, there are many commercial rating engines available in the market for calculating rates for motor insurance policies. These rating engines can be customized with minimal effort according to an organization's requirements. As multiple solutions are involved with this process, cross-platform computability was perceived as a major challenge to this solution. However, wider use of Service Oriented Architecture in the past decade has enabled applications in different platforms to interact with each other through Web Services.

Problems Associated With Testing Soa

Testing an application implemented in this framework is a tough task. Any tester will readily agree to the fact that the most redundant but necessary phase during testing is Test Design. There are four major reasons that will support this argument,

- Test design involves a significant part of the overall testing effort.
- In case of a change of requirements, the designed test artifacts need to be revisited to incorporate the change.
- Effort is involved in maintaining the pack.
- In most cases the test design artifacts become obsolete once test execution is completed.

All these arguments are infinitely compounded when testing a SOA based application. Testing SOA is typically considered as a front-loaded process. Contrary to the general testing process, where test execution consumes the major portion of the overall effort, test execution in SOA is a relatively straightforward process. With the help of many tools available in the market, test execution occupies less than 30% of the overall testing window. Invariably, however, test design for SOA is a much more complicated process. Often testers obtain the schema from either the base XML or WSDL of the component, extract the different nodes in the XML, validate the input and output values, and finally apply parameterization only to find that the base XML has been changed overnight. It is a never ending, vicious cycle, with testers being kept in the dark until a few days before commencement of test execution or in some cases on the day of test execution. Therefore testers are ill equipped to start with test execution immediately and spend most of their time in updating the test design according to the latest requirements. This is further complicated when the protocols for interaction between different entities undergo frequent changes.

Application Of MBT

Model based testing is a viable solution to solve this problem. It can be applied at two different levels in the architecture to neutralize the impact of constant changes in the application. It can be applied to address the problem of traversal across the entities, and also in testing each individual entity.

Tier 1 Implementation – Traversal Across Entities

Typically, any application involves business rules that govern how the user can navigate through the application. For example, in the aforementioned motor insurance scenario, the system should not process payments until the user has traversed through a single or a multiple list of entities previously. For simple applications, the manual identification of such valid business traversal across entities is a straightforward process, but it becomes a tedious one for complex applications, and responding to changes in requirements is also a time-intensive task.

By representing the modules in the form of a directed cyclic graph, model-based testing can be applied to identify the valid business graph. This can be implemented by converting the single directed cyclic graph into multiple directed acyclic graphs. For example, the motor insurance scenario could be represented as shown in Figure 2.

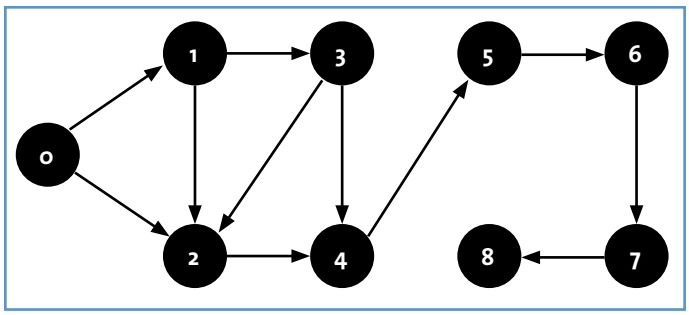


Figure 2. Sample Directed Acyclic Graphs Representing Motor Insurance Policies

Once the application flow is specified in the above manner, there are many tools and techniques available to find all the valid application traversal paths. For an application that enforces limited business rules, statistical methods, such as Sequential Polling Algorithm and Modified Condition & Decision Coverage (MCDC), can be applied to arrive at the list of valid paths. Similarly, for a complex application that contains multiple business rules, Dijkstra's algorithm could be applied using either depth-first or breadth-first search depending on the nature of the business problem. There are many tools that are available in the market to implement application traversal paths once the graph is created. In case of change in the business process, the user needs to update the graph to automatically repopulate all the transactions.

Tier 2 Implementation – Testing Individual Entities

Once the various traversal paths have been established, model-based testing can be applied to test each entity on its own. In this case each entity will consist of a Base XML or a WSDL file. Applying the techniques of depth-first algorithm, the XPath and XQuery of each individual node in the WSDL can be extracted. Once the values are extracted, we can either apply a brute force algorithm to arrive at all the possible permutations and combinations, or use tools to apply refined techniques like Orthogonal Array Testing (OAT) to arrive at an optimized set of values.

For example, let's assume that the Rate Engine entity in the motor insurance application can be denoted by linear equation,

```
R Rate = (((2x + 4y) * 3z) - 4p)
where x = {1,2,3}; y={8,9}; z={6,7}; p={0,1}
```

Applying the Orthogonal Array Testing at third degree, the overall number of combinations can be reduced from 24 to 14, which is represented in Figure 3.

x	y	z	p	R_{Rate}
1	9	7	1	**794**
2	8	6	0	**648**
3	9	6	1	**752**
1	8	7	0	**714**
2	9	7	0	**840**
3	8	7	0	**798**
2	8	7	1	**752**
1	8	6	1	**608**
1	9	6	0	**684**
3	8	6	0	**684**
3	9	7	1	**878**
2	9	6	1	**716**
3	9	7	0	**882**
3	8	7	1	**794**

Figure 3. Sample Implementation of 3rd degree of OAT

Now the above combinations can be automatically parameterized to the Base XML's to validate the output of the service. This output can then be used in conjunction with the different traversal paths derived in the Tier 1 step to automatically create the test design for the application.

Automatic Execution Of Test Scenarios

One of the key aspects of performing model-based testing is the automatic execution of the test scenarios. Once the Tier 1 and Tier 2 activities have been completed, there are many tools available in the market that can be configured to perform automated execution of the scenarios. These tools also perform various kinds of reporting.

In a nutshell, once the system has been segregated into separate models, MBT can be applied to find the different types of interaction and also validate each individual model to produce the test design artifacts. These artifacts can then be configured to run against the various services according to the different paths.

Conclusion

Model based testing as a concept has been in discussion for a long time, but has always faced challenges during the implementation process. This has been due to many underlying factors. However, the time is now ripe for implementing model-based testing, especially for solving problems in fast growing technologies like SOA. The two-tier approach to model-based testing of service oriented architecture described in this article empowers us to utilize the full potential of both processes, resulting in performing faster and better testing, which is the need of the hour.

> biography

Prasad Ramanujam
currently works as Project Manager at Cognizant Technology Solutions, Chennai. Being an 11 year veteran in the IT industry and a recognized leader in agile methodology, he is a frequent speaker at industry conferences on the subject of successfully implementing agile testing techniques in organizations. He has supported many multi-national clients in various industries in becoming high-performing businesses by establishing a structured and strategic approach to Quality Assurance.

Vinothraj Jagadeesan
holds a degree in Computer Application from the University of Madras and has extensive testing experience in niche areas including Open-Source Test Automation, Testing SOA and Agile Implementation across different locations. Having successfully completed more than 9 certifications, he is an insurance domain expert, and is certified in both the US and the UK by AICPCU and CII respectively. He is also an HP certified professional in both Quality Center and QuickTestPro. Currently, he oversees the testing for a leading specialist insurer in the UK in the area of implementing a solution using Scrum-of-Scrum.

Venkatesh Ramasamy
is an ISTQB® certified testing professional working as Senior Test Analyst in Cognizant Technology Solutions, Chennai. He has excellent experience in managing enterprise IT project life cycles through all testing phases. He has developed many software products for doing end-to-end test management activities, thus optimizing testing costs and improving the quality of the application. He has presented around 16 research papers in various technology fields, such as embedded systems, microelectronics, communication, information technology, etc.

Certified Agile Tester

Pragmatic, Soft Skills Focused, Industry Supported

CAT is no ordinary certification, but a professional journey into the world of Agile. As with any voyage you have to take the first step. You may have some experience with Agile from your current or previous employment or you may be venturing out into the unknown. Either way CAT has been specifically designed to partner and guide you through all aspects of your tour.

The focus of the course is to look at how you the tester can make a valuable contribution to these activities even if they are not currently your core abilities. This course assumes that you already know how to be a tester, understand the fundamental testing techniques and testing practices, leading you to transition into an Agile team.

The certification does not simply promote absorption of the theory through academic mediums but encourages you to experiment, in the safe environment of the classroom, through the extensive discussion forums and daily practicals. Over 50% of the initial course is based around practical application of the techniques and methods that you learn, focused on building the skills you already have as a tester. This then prepares you, on returning to your employer, to be Agile.

The transition into a Professional Agile Tester team member culminates with on the job assessments, demonstrated abilities in Agile expertise through such forums as presentations at conferences or Special Interest groups and interviews.

Did this CATch your eye? If so, please contact us for more details!

Book your training with Díaz & Hilterscheid!

Open seminars:

23.04.12–27.04.12 in Berlin, Germany
(German tutor and German exam)

19.03.12–23.03.12 in Amsterdam, The Netherlands
11.06.12–15.06.12 in Stockholm, Sweden
02.07.12–06.07.12 in Orlando, USA

Díaz Hilterscheid

Díaz & Hilterscheid GmbH / Kurfürstendamm 179 / 10707 Berlin / Germany
Tel: +49 30 747628-0 / Fax: +49 30 747628-99
www.diazhilterscheid.de training@diazhilterscheid.de

Analyzing decision tables

by Alon Linetzki

Decision tables are one type of analysis method (or technique), that is commonly used by developers (for design and specification documents), and by others in software engineering and other disciplines – e.g., by test engineers, test analysts, test managers, etc.

Decision tables are used mainly because of their visibility, clearness, coverage capabilities, low maintenance and automation fitness. Also decision tables are a useful source of information for model-based testing, and work well on rule based mechanisms or modeling techniques.

In the article I describe my understanding of how to use and analyze decision tables in a way that will give us the most benefits from them, and with the aim of constructing good test cases or test scenarios later on from that analysis.

Please note that this analysis has been a great experience for me in my context, and you should check it out in yours...

What are decision tables?

Decision tables provide a useful way of viewing and managing large sets of similar business rules.

Decision tables are composed of rows and columns. Each of the columns defines the conditions and actions of the rules [1].

We can look at decision tables as a list of causes (conditions) and effects (actions) in a matrix, where each column represents a unique rule.

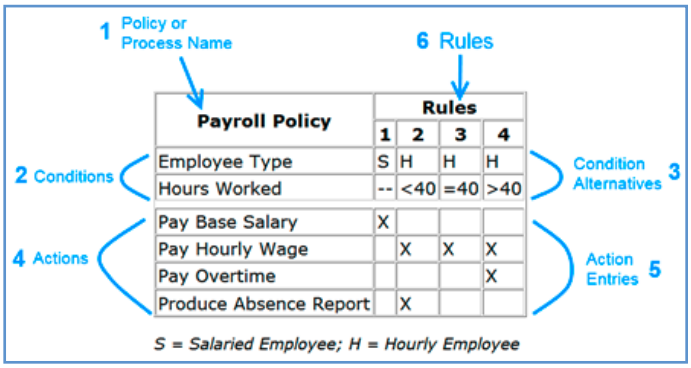


Figure 1.0 Structure of a Decision Table [3]

The purpose is – as commonly used – to structure the logic for a specific specification feature.

One can add new rows to a decision table and fill in its cells to create new rules. When the rules are executed, if the conditions are met, the actions in that column are performed. One can also add preconditions that apply to all the rules.

Types and forms of decision tables

Beside the basic four-quadrant structure, decision tables vary widely in the way the condition alternatives and action entries are presented.

Some decision tables use simple true/false [Y/N or 1/0] values to represent the alternatives to a condition (similar to if-then-else), other tables may use numbered alternatives (similar to switch-case) [2].

In a similar way, action entries can simply represent whether an action is to be performed (for this we mark the actions to perform), or in more advanced decision tables, the sequencing of actions to perform (for this we number the actions to perform).

The limited-entry decision table is the simplest to describe. The condition alternatives are simple Boolean values, and the action entries are check-marks, representing which of the actions in a given column (rule) are to be executed [2].

Example: Suppose a technical support company writes a decision table to diagnose printer problems based upon symptoms described to them over the phone from their clients.

The following is an example for a balanced decision table [2]:

		Rules							
Conditions	Printer does not print	Y	Y	Y	Y	N	N	N	N
	A red light is flashing	Y	Y	N	N	Y	Y	N	N
	Printer is unrecognised	Y	N	Y	N	Y	N	Y	N
Actions	Check the power cable			x					
	Check the printer-computer cable	x		x					
	Ensure printer software is installed	x		x		x		x	
	Check/replace ink	x	x		x	x			
	Check for paper jam		x		x				

Figure 2.0 Limited-entry decision table

There are, however, other types of decision table.

In the following example the condition alternatives are not Boolean values, and the actions should be combined to form the complete action as a result from that unique choice of values used in the conditions. It is called *Extended-Entry* table.

In an Extended Entry table, the conditions are written as more open ended questions such that part of the condition extends itself into the condition entry. This can be applied for both the input values (conditions) and the output values (actions). Instead of writing a Y/N condition, we write an open-ended one, that the answer is the age itself [3]:

Residence Assignment	Rules		
	1	2	3
Student Age	<21	<21	>=21
Residence Type	Co-End	Non Co-Ed	--
Residence Assigned	Trudeau	MacDonald	Laurier

Figure 3.0 Extended-entry Decision Table

The third type of decision tables are the *Mixed-Entry* tables, which have both the Boolean and the non-Boolean entries:

Conditions:	Values:	Rule 1	Rule 2	Rule 3	Rule 4
Age range	18–25, 26–35, 36–52	18–25	18–25	...	...
Parking place	Street, Garage, Mix	Street	Street	...	...
Horse Power <=50	Yes, No	Yes	No	...	...
50<Horse Power <=100	Yes, No	No	Yes	...	...
Actions:					
Basic Premium	400 Euro	X	X	...	...
Parking	-5%, +5%	+5%	+5%	...	...
Horse Power <=50	Basic Premium	400		...	...
50< Horse Power <=100	BP + 100 Euro		500	...	...
Age 18–25	+10%	+10%	+10%	...	...
Age 26–35	Basic Premium			...	...
Age 36–52	-5%			...	...
Total premium		400 + 15%	500 + 15%	...	...

Figure 4.0 Mixed-entry Decision Table

Of course, we can find more types of decision table, which I will not analyze or present in this article.

How to read and analyze decision tables

In order for us to utilize the full potential of decision tables, I suggest we follow a simple process for understanding and analyzing them.

I assume we get those decision tables from development, and they are included in the specification documents. Otherwise, I usually construct them myself from the specifications.

Analyzing decision tables:

Step – 1: Clearly understand the decision table

In this step, we analyze the symbols, graphics, letters and numbers of the decision table and make sure we understand them all clearly.

For example, the symbol '-' may be used for either 'Not-relevant' or as 'Not-applicable', or may have some other meaning.

See the following example.

Conditions:	Values:	Rule 1	Rule 2	Rule 3	Rule 4	Rule 5	Rule 6
Valid Credit Card	Yes, No	No	Yes	Yes	Yes	Yes	Yes
Valid Card PIN	Yes, No	-	No	No	Yes	Yes	Yes
Invalid PIN, 3rd time	Yes, No	-	No	Yes	-		
Valid Amount	Yes, No	-	-	-	No	Yes	Yes
Credit Limit Verified	Yes, No	-	-	-	-	No	Yes
Actions:							
Reject the Card	Yes, No	Yes	-	-	-	-	-
Renter PIN	Yes, No	-	Yes	-	-	-	-
Renter Amount	Yes, No	-	-	-	Yes	Yes	-
Give out Cash	Yes, No	-	-	-	-	-	Yes
Eject Card	Yes, No	-	-	-	-	-	Yes
Keep Credit Card	Yes, No	-	-	Yes	-	-	-

Figure 5.0 ATM – meaning of '-'

In the above simple ATM, the '-' means 'Not Relevant'. However, we can clearly say that is not the case in the following decision table:

Conditions:	Values:	Rule 1	Rule 2	Rule 3	Rule 4	Rule 5	Rule 6	
Shipment?	Yes, No	Yes	-	-	-	-	-	-
Wrapping?	Yes, No	-	Yes	-	-	-	-	-
BD Card	Yes, No	-	-	Yes	-	-	-	-
Gift type (Age)	1–8, 9–12, 13–17, 18–24	-	-	-	1–8	9–12	13–17	18–24
Actions:								
Shipment Charge	10 Euro	+10 Euro	-					
BD Card Fee	7 Euro	-	-	+7 Euro				
Wrapping Fee	3 Euro	-	+3 Euro					
Gift Type	A, B, C, D	-	-		A	B	C	D
Handling Fee	0.5, 1, 1.5, 2 Euro	+2 Euro	+1.5 Euro	+1 Euro	+0.5 Euro	+1 Euro	+1.5 Euro	+2 Euro

Figure 6.0 Purchase a present – different meaning of '-'

Here, the '-' is different based on its location. In the first 3 conditions, it means the opposite from what is written in the same line (i.e. for Shipment, in Rule 1 it is 'Yes', whereas for Rule 2, it means 'No'; in the Gift type condition, it means 'Not relevant').

The '+' on the prices or fees means that the fees are accumulative (thus there is probably a missing 'Total Fee' line in the actions).

Step 2 – Identify missing combinations

After realizing the decision table symbols, we should be looking for missing combinations, which make sense in our context.

Some combinations are left out as they are not relevant for the context (i.e. using 'Unix' as OS together with 'IE browser', or using a 'Married' status after a status of 'Dead'), but some are just not designed by the system engineer or other professional who created the decision table.

When we find such cases, we perform a 'What-If' discussion to find out if this combination is relevant, and then pass it to the system engineer or author of the document for evaluation.

Step 3 – Identify dependencies between separate rules

In order for us to be able to identify the logic behind the table, we must determine the existing dependencies between the different rules in the decision table.

There are many different types of dependency:

- Can run in parallel
- Can run only uniquely at same time
- Can run only after another rule has been executed
- Can run only for the first time (then blocked/suspended)
- Can run only once (then blocked)
- etc.

In the next example, we review the previously used ATM table (Figure 5.0), and analyze dependencies. We can clearly see that Rule 2 must happen twice before Rule 3 can be applied. So only after having entered an invalid Card PIN twice, can we reach the third time.

This kind of analysis can assist and support us in scheduling our tests, or breaking them down from "End-to-End" into smaller ones for execution and low maintenance. If we run Rule 2, testing it first, then run Rule 3, we will not just be able to identify possible

defects in Rule 2 operation, but will also not need to simulate the situation of entering the wrong PIN twice for running Rule 3.

In Figure 6.0, we can see that the dependency of the first three conditions (Shipment, Wrapping, BD Card) is that they can run at the same time (in parallel), which is applied for rules 1-3, while the last condition (Gift type), can only happen uniquely (so selecting only 1 value of age from the ranges is possible), which is applied for rules 4-7.

Step 4 – Identify and apply other techniques on condition values

Once we have identified the dependencies, we can already identify ways of producing fewer test cases, starting with identifying other techniques that can be applied on the decision table, such as Equivalence Class Partitioning (EC), and Boundary Value Analysis (BVA).

When looking at Figure 6.0, we can clearly see ranges (Gift Type) that can be used for BVA for those within the ECs. We can identify values such as 1, 8, 9, 12, 13, 17, 18, and 24 (some serving as internal boundaries), and also boundaries such as 0 and 25 (which are external boundaries). Negative values can also be linked, such as -1 and 100 (3 digits).

Gift type (Age)	1–8, 9–12, 13–17, 18–24	-	-	-	1–8	9–12	13–17	18–24

Figure 7.0 Gift Type – boundaries and equivalent classes

Step 5 – Use your head – Optimize what's left...

We continue to analyze the table in Figure 6.0. If asked: *"How many test cases should we plan to test this decision table?"*, one can calculate this in the following way:

Rules 1-3 can run in parallel, they are Boolean, so this means 23 test cases (=8). Rules 4-7 can only run uniquely, and have (together with the EC and BVA analysis, plus negative cases) 12 options. That is 8 x 12 = 96 test cases for the combinations.

However, when we try to test the Gift Type, do we really want to test that against all 8 combinations of rules 1-3? Or should the orientation focus on rules 4-7? The latter is the answer of course.

So we can take only 2 out of the 8 combinations of rules 1-3, say the one that gives us minimal fee and the one that gives us the maximum fee, and use only those to combine with the other values from rules 4-7.

The calculation now yields the following: 23 + 2 x 12 = 8 + 24 = 32.

What we have done is trying to figure out the risks of dropping some combinations, and optimizing the number of test cases to be executed, in order to achieve coverage and a variety of cases. With a little more risk taken (in my subjective view), we have reduced the number of test cases from 96 to 32, which is a reduction to 33%!

Even when using strong techniques, it is essential to analyze this kind of optimization for getting a better balance between cost and risk.

We can see the same thing in a simpler example when looking at Figure 5.0, the ATM. This time, however, we would like to enhance the number of test cases.

In this case, the condition Valid Credit Card is in fact an EC, as the result is the same for various kinds of non-valid card values (e.g. expired date, stolen card, magnetic tape not working, etc.). Although the definition of EC states that one representative from the EC is enough, we would like to test some more cases as our customers in real life will! So we would recommend testing a few of those for the Rule 1 column, but also use boundary test cases on the Credit limit and on the amount entered.

How to turn decision tables into test cases

One simple logical way is to treat every rule as a test case, and cover them (at least once) in our test cases.

In general, the condition values used in a unique way in each rule (causes) will serve us as the different inputs for the test cases, and the values (effects) used in the Actions will serve us as the expected results.

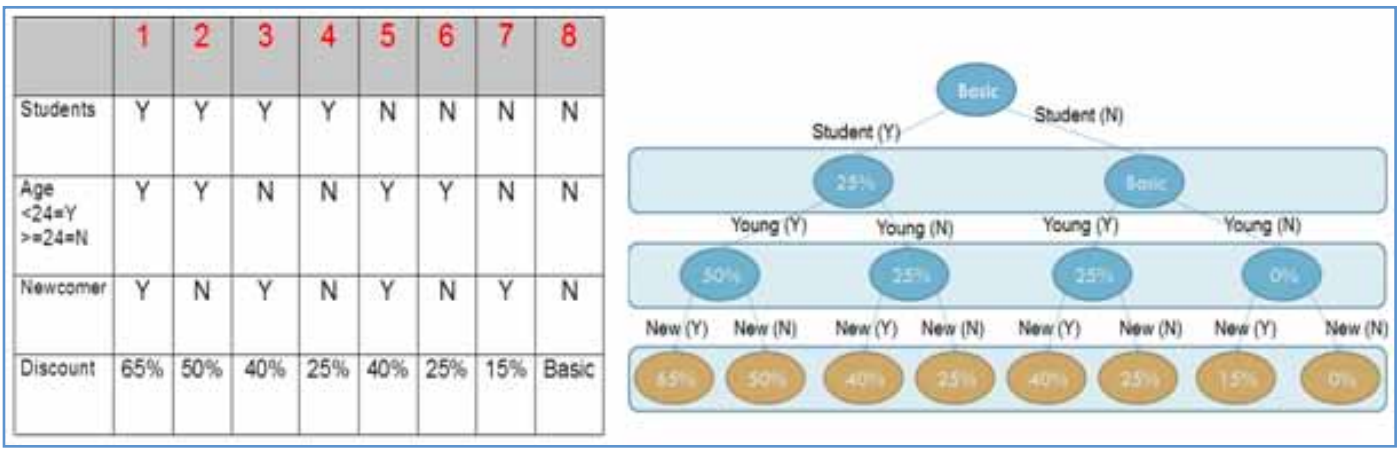


Figure 8.0 Example for a Decision Table vs. a Decision Tree

However, we can look at large sets of combinations and large tables in a different way. We can use the *decision tree* technique to transform those into a more visual way to 'see' our test cases, or we can construct the whole conditions/actions into a decision tree right from the start.

In the following example, three conditions influence a discount policy: Student (= 25% discount), Age <24 (= 25% discount), Newcomer (up to 1 year subscription = 15% discount). Discounts are accumulative. If no discount is valid, the Basic price is paid.

	Inputs	Expected results
Test 1	Student at the Age of 23, which is Newcomer	Student gets a 65% discount on Basic price
Test 2	Student at the Age of 19, which is NOT a Newcomer	Student gets a 50% discount on Basic price
Test 3	Student at the Age of 31, which is Newcomer	Student gets a 40% discount on Basic price
Test 4	Student at the Age of 39, which is NOT a Newcomer	Student gets a 25% discount on Basic price
Test 5	a person (NOT a student) at the age of 18, which is a Newcomer	Person gets a 40% discount on the Basic price
Test 6	a person (NOT a student) at the age of 22, which is a NOT Newcomer	Person gets a 25% discount on the Basic price
Test 7	a person (NOT a student) at the age of 45, which is a Newcomer	Person gets a 15% discount on the Basic price
Test 8	a person (NOT a student) at the age of 62, which is a NOT Newcomer	Person gets no discount on Basic price

Figure 9.0 Test cases from Decision Table

Of course, we should add boundary test cases, for cases just before the age of 24 (=23), at 24 and just beyond the age of 24 (=25):

	Inputs	Expected results
Test 9 (B)	Student at the Age of 23, which is Newcomer	Student gets a 65% discount on Basic price
Test 10 (B)	Student at the Age of 24, which is Newcomer	Student gets a 50% discount on Basic price
Test 11 (B)	Student at the Age of 25, which is Newcomer	Student gets a 40% discount on Basic price

Figure 10.0 Boundary Test cases from Decision Table

Summary

Using a simple process of learning about the decision tables, understanding them, and analyzing them using different techniques in order to reduce the number of test cases, can yield in a very well balanced number of test cases, or also a very low number. These can then be used to produce those for the test design phase.

The process in a nutshell:

- Step 1: Clearly understand the decision table
- Step 2: dentify missing combinations
- Step 3: Identify Dependencies between separate rules
- Step 4: Identify and apply other techniques on condition values
- Step 5: Use your head – Optimize what's left...

The two major points to remember are that finding the dependencies between rules can help us schedule our tests better and smarter, and that optimizing the number of test cases after calculating the number of tests cases using the techniques can help us a lot in a further reduction of test cases, while taking only a small additional risk.

Good luck!

References

[1] IBM Websphere ILOG Rules – http://publib.boulder.ibm.com/infocenter/brdotnet/v7rom3/index.jsp?topic=%2Fcom.ibm.websphere.ilog.brdotnet.doc%2FContent%2FBusiness_Rules%2FDocumentation%2F_pubskel%2FRules_for_DotNET%2Fps_RFDN_Global743.html

[2] Decision Table, Wikipedia – http://en.wikipedia.org/wiki/Decision_table

[3] Sheriden Academic Server – http://www-acad.sheridanc.on.ca/~jollymor/info16029/week4.html#format

> biography

Alon Linetzki
has been a coach and consultant in the fields of testing and quality assurance for the last 28 years. He started his career as a SW developer, SW development leader, system analyst, and system administrator before becoming a tester.
Alon has coached and consulted more than 1,000 professional engineers and managers in more than twenty companies so far.
His passion is about improvement, and he is keen on people, how we think, how well we work, how we can do better. He has been an Effective Personal Communication coach for 3 years, and shares this experience with agile and other teams.

Alon established the Israeli branch of the ISTQB® - ITCB (2004), served as Vice President and marketing director, and leads the ISTQB® Partner Program worldwide. He also established the SIGiST Israel – Israeli Testing Forum (2000).

The ASQF Career Center

We've got the right job for you.

Examination Evaluation (m/f) (Certified Agile Tester®) iSQI GmbH
various sites

Invigilation and Evaluation of Exams (m/f)
iSQI GmbH
Munich, Stuttgart, Constance

Software Engineer (m/f) - Java
infoteam Software AG
Dortmund, Erlangen and Zurich

Software Engineer (m/f) - C#, .NET
infoteam Software AG
Dortmund, Erlangen and Zurich

Software Architect (m/f)
infoteam Software AG
Dortmund, Erlangen and Zurich

Senior Consultants IT Management & Quality Services (m/f) Díaz & Hilterscheid Unternehmensberatung GmbH, Berlin

Consultant - Specialist for SCM, ALM and Test Automation (m/f) elego Software Solutions GmbH, Berlin and Dresden

Clerk (m/f) Exam Evaluation / Office Management iSQI GmbH
Potsdam

Software Tester (m/f)
ckc ag, Braunschweig

Trainee Test Specialist / Junior Consultant Test (m/f)
ckc ag, Braunschweig

Quality Assurance Engineer / Software Tester test automation (m/f)
Sedo GmbH, Cologne

Technical Officer Telecommunications (m/f)
IABG mbH, Ottobrunn

Consultant Digital BOS Radio (m/f)
IABG mbH , Ottobrunn

Software Engineer (m/f)
ERGO Direkt Versicherungen
Nuremberg

Consultant and Trainer (m/f)
SOPHIST GmbH
Nuremberg

Lead Test Manager (m/f) at Healthcare Syngo
Siemens AG, Erlangen

Basis Software Engineer LIN (m/f)
Brose Fahrzeugteile GmbH & Co. KG
Coburg

Software Engineer (m/f) AUTOSAR
Brose Fahrzeugteile GmbH & Co. KG
Coburg

Consultant Test Data Management (m/f)
GFB Softwareentwicklungsgesellschaft mbH, Oberursel

Software Engineer (m/f) .NET (focus on Q-uo Product and Solution Engineering) GFB Softwareentwicklungsgesellschaft mbH, Oberursel

Software Tester (m/f) Quality Assurance
GFB Softwareentwicklungsgesellschaft mbH, Oberursel

Software Engineer (m/f) .NET
GFB EDV Consulting und Services GmbH, Oberursel

Detailed information on all job offers on www.asqf.de/stellenangebote.html *(German only)*

Assessing data quality

by Harry M. Sneed & Richard Seidl

1. The loss of data quality

The existing databases of most IT users are the product of a long evolution. It began with their conception by a database designer on the basis of the information available at that time. This was followed by the hand coding or the automatic generation of a database schema reflecting the design. After that the database was installed and applications began to use it. As new applications came along, the structure was altered or extended to accommodate them. These structural changes were often made in an ad hoc manner, so that the complexity of the structure grew and the quality sank, much like software systems according to the laws of software evolution [1]. In addition to this structural deterioration, the contents of the database became corrupted by erroneous programs storing incorrect values and deleting essential records. With large databases it is difficult to recognize such quality erosion, but over time it spreads like a cancerous infection causing more and more system failures. Thus not only program quality but also data quality suffers from erosion [2].

1.1. Failure to foresee data evolution

At the beginning of a database life cycle the designer is obliged to make assumptions about the data quantity and the way the data will be used. He must also foresee how the data will evolve. The design of the database is based on such assumptions. The same assumptions are carried over into a hand coded database schema in which the tables are defined and linked to one another by foreign keys and indexes. If the assumptions prove to be right, the database can evolve with no problems. If not, the database structure will soon be obsolete and the form will no longer fit to the content. It will also become increasingly difficult to adapt. If the data volume grows much more than was expected and the data usage turns out to be quite different from what was originally foreseen, it becomes necessary to reengineer the database [3].

1.2. Lack of data independence

Often enough database designers cannot predict the actual growth of a database, nor can they foresee how it will really be used. When they first design the database, they are likely to design it solely from the viewpoint of a single application and not from a global viewpoint of all possible applications. Thus, from the beginning, the data is not independent of the applications using it. Later the structure has to be adapted to accommodate more and different applications, but the further the design is stretched, the more brittle it becomes. At some point it is no longer adequate for any of the applications [4].

1.3. Violation of the normal forms

Each provisionary patch to the database structure has negative consequences. Instead of creating new sub-tables when additional attributes are required, the local database administrator will simply add them to existing tables. As a result, the rows of the databases become increasingly longer and harder to handle. When new keys are needed, they are added as indexes. As a result there are more and more indexes. Data groups are added to existing rows, thereby violating the second normal form, and repetitions of the same data items are made in order to avoid creating new sub-tables, thus violating the principle of first normal form. Additional foreign keys are added which increase the number of dependencies between tables, and sub-keys are added to the primary key, which often do not pertain to the rows as a whole. This amounts to a violation of the principle of third normal form. In the end the database no longer conforms to the normal forms for relational databases with a subsequent loss of quality and an increase of complexity [5].

1.4. Misuse of stored procedures

Another feature contributing to a loss of portability and data independence is the use of stored procedures. If stored procedures were only used for access operations and to ensure the integrity of the data, it would be acceptable. However, they are often misused to perform processing of logic. Rather than putting business rules in separate rule modules or placing checking routines in the client components, developers hide them in the stored procedures where they are no longer visible to the tools processing the programs. Stored procedures full of selections and loops, sets and declares, are a definite sign of misuse. They bind the database to a particular application [6].

1.5. Inconsistent and incorrect data updates

The contents of a database suffer not only from errors when executing the programs but also from inconsistent changes. The type of a data attribute contained in two different tables may be changed in one table but not in the other, or the value of an attribute is no longer compatible with the type of that attribute in the using program. The Frankfurt Stock Market once had to be closed down for half a day because of an invalid value in the database leading to a data exception error in the processing job [7]. Unfortunately, there are much more bad data in existing databases than users assume, and this bad data can at any time lead to a system crash.

Another problem with the content of databases is that of missing and redundant records. Missing records are those which are still referenced, but which have been either intentionally or unintentionally deleted. They may have been deleted by one application, but are still needed by another. Redundant records are those which should have been deleted, but for some reason were not. They remain as corpses in the database occupying valuable space, even though they are no longer used [8].

The quality deficiencies noted here are only a subset of the many quality problems with application databases. Low database quality has become a major concern for IT users and is the motivation for the three audit processes presented in this article:

- analyzing the database schema
- measuring the database structure
- validating the database content

2. Statically analyzing database schemas

The database model needs to be statically analyzed to detect quality deficiencies as well as to measure the size, complexity and quality of the database structure. To this end, an automated auditing process such as the one depicted in Figure 1 is required (see Fig. 1: Static database analysis).

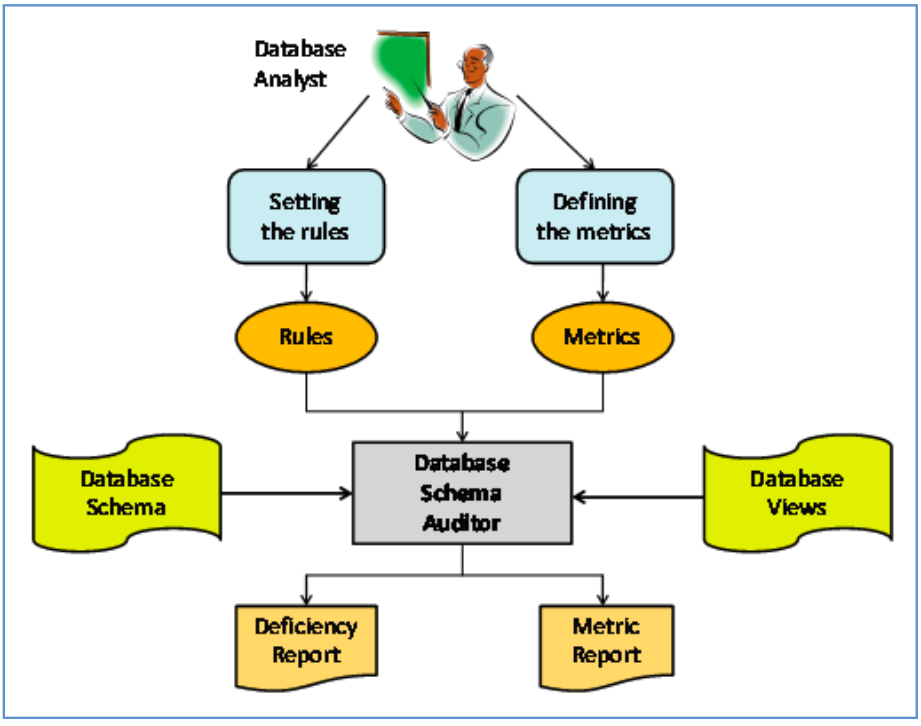


Figure 1: Static Database Analysis

The first step in auditing a database is the static analysis of the database schema with the goal of detecting rule violations. Just as with program code, database schema code needs to be governed by rules to enforce good coding practice [9]. The rules are, on the one hand, directed toward the fulfillment of certain data quality goals such as security, compatibility, performance and portability. On the other hand, they are intended to make it easier to change and extend the database.

In order to assign rule violations different weights, it is necessary to classify them. The database auditor distinguishes between major, medium and minor rule violations. Major rule violations like violating the first normal form are a major barrier to the further development and re-use of the data. They may also cause failures. Medium rule violations like exceeding a given size limit reduce the modularity and maintainability of the database. Minor rule violations such as naming convention violations make it difficult to maintain the data and keep it consistent with the code. Major deficiencies are weighted by 2, medium ones by 1 and minor ones by 0.5.

2.1. Checking vendor specific features

There are many features of a database which are vendor specific, i.e. they are not standard SQL. They should be used with care. Some options should be forbidden because they effect performance or deter the transfer of the data. Others, like the NULL option, should be mandatory for the same reasons. It is up to the responsible database analyst to identify which features should be prohibited and which should be mandatory.

2.2. Checking the normal forms

Avoiding vendor specific features may lower performance, but it makes the database more independent of the vendor, i.e. the data becomes more portable. Having a primary key in every record may not be required by the SQL language, but it can be a rule to be enforced. Having a foreign key is also not necessary, but if the table is dependent on another table, at least one should be defined. In no case should a table have repeated data of the same type or sub-groups of data. Such obvious violations of the normal forms can be recognized and identified as bad practice by means of static analysis [10].

2.3. Setting limits

There may also be limits as to how many attributes are allowed in a set of data and how long a row may be. It is necessary to check that these limits are not exceeded. Furthermore, it may be desirable to forbid certain data types such as packed-decimal, binary, floating-point or time-stamp data for the sake of compatibility with other databases of another vendor. Incompatible data types are one of the leading causes of errors in data migration [11].

In summary, one can conclude that there should be a set of rules for designing databases and that these rule should be enforced. Typical rule viola-

tions are:

- unique identifier is missing in table
- foreign key is missing in dependent table
- table contains a repeated data attribute
- table contains a sub-group
- table has no external view defined
- table has no indexes
- primary key contains too many sub-keys
- NULL option is missing
- DELETE option is missing
- table has an incompatible data type
- number of attributes exceeds maximum limit
- length of row exceeds maximum allowed length
- schema is insufficiently commented

It is the task of the schema-auditing tool to check such rules and to report their violations. It is then the job of the database administrator to correct the database schema accordingly.

3. Measuring the database structure

To evaluate the architecture of a database, it must first be measured in order to base the evaluation on solid numbers. Without numbers it is not possible to compare, and without being able to compare, it is not possible to judge [12]. The measurements of an existing database can be compared with those of a benchmark database or simply with those of another database. Data measurements can be simple counts, or they can be complex metrics for size, complexity and quality [13].

3.1. Database size metrics

As for the rule checking, a tool is needed to perform the measurement, to count the database attributes and to compute the metrics. There are several database attributes to be counted. In an SQL database they include:

- lines of SQL code
- number of tables
- number of attributes
- number of keys
- number of views
- number of relationships
- number of stored procedures
- number of stored access operations
- number of stored procedure statements
- number of integrity rules
- number of stored procedure rules
- number of test cases required to cover the database

From these measures two size metrics are computed:

- function points
- data points

The function points of a database are either 15, 10 or 5 depending on the number of attributes and keys [14]. Databases with over 50 attributes or 2 keys get 15 function points, databases with 20 to 50 attributes or more than one key get 10 function points, databases with less than 20 attributes get 5 function points. The data points are 1 for each attribute, 2 for each key and index, and 4 for each view [15]. These metrics are used to predict the costs of evolution projects that involve changes to the data.

3.2. Database complexity metrics

In addition to the size metrics, complexity and quality metrics should be computed. All metrics are ratio metrics derived from a relation of one set of attributes to another. They are computed using algorithms for comparing actual values to benchmark values based on empirical studies of similar databases. For each complexity metric there is a lower and an upper bound. The median complexity is the geometric mean of the two boundary values. The six computed complexity metrics are:

- Content complexity as the relation of data types and data keys to the number of data attributes
- View complexity as the relation of views to the number of tables
- Access complexity as the relation of access paths to the number of access object views and tables
- Relational complexity as the relation of tables to table relationships to the number of tables
- Structural complexity as the relation of structural elements (tables, views and indexes) to the number of basic elements (attributes and keys)
- Storage complexity as the relation of attributes stored to the size of the storage area in bytes

The weighted average of these six complexity metrics gives the overall complexity of the database on a ratio scale of 0 to 1. The complexity ratio should be as low as possible [16].

3.3. Database quality metrics

The desired qualities of a database have been propagated in the literature on this subject, in particular by J-L. Hainaut [17], Peter Aiken [18] and Michael Blaha [19]. These and other authors emphasize the importance of data independence, data flexibility, access ability and data portability. One of the greatest detriments to software migration is the intertwining of business logic with access logic. One of the greatest barriers to data evolution is the inflexibility of the database structure. During the 1980's both Dominique Warnier [20] and Michael Jackson [21] propagated data driven software design, a method by which the program structure is derived from the data structure. The results were programs that reflected the current hierarchical and networked data models. When it became time to move to relational databases, the program structures no longer fitted. They had to be reengineered at a high cost. What was once a good idea to come up with a quick solution, turned out to be the greatest barrier to system modernization. Even today the structure of many relational databases is based on the requirements of the original application. It was not foreseen how the data could be used in the future [22].

The solution to the data independence problem is that each application should have its own views of the data. Views were defined to make this possible [23]. Of course, this increases the complexity of the database, but there are always trade-offs to be made. Here the trade-off is between database independence and data complexity. The same applies to data accessibility which requires more keys and more indexes to be able to access the data in many ways. Data flexibility implies that the database can be easily changed and extended. For that, there should be many smaller tables rather than fewer big ones. However, like with object-oriented programming, this causes more relationships between the tables and with that higher complexity [24]. Storage efficiency means storing more data in less space, but that too increases complexity. This shows that there is hardly any way to

bring up the quality of a database feature without causing higher complexity or sacrificing the quality of another feature. This is a dilemma which database designers have to resolve. Attaining high quality, while maintaining low complexity is an optimization problem which depends on the context in which the data is used. If data is used in different contexts, a best possible compromise has to be found.

For each quality metric there is also a lower and an upper bound derived by comparing various databases. The median quality is the geometric mean of the highest and lowest quality values. The six computed qualities are:

- data independence as the relation of data views and indexes to the number of data tables
- data accessibility as the relation of access paths to the number of stored attributes
- data flexibility as the relation of tables, keys and indexes to the number of data attributes
- data storage efficiency as the inverted relation of stored attributes to the size of the storage area in bytes
- data conformity as the relation of rule violations to the number of schema lines

The weighted average of these quality metrics gives the quality coefficient for the database as a whole. It should be as high as possible. According to the quality measurement scale of ISO standard 9126 scores above 0.8 are excellent, from 0.6 to 0.8 good, from 0.4 to 0.6 satisfactory and under 0.4 unsatisfactory [25].

4. Validating the database content

Validating the content of a given database requires an oracle [26]. There should be someone or something that declares what should be in the columns of a table in general and in what line in particular. This oracle could be a person, a program, an interpretable specification or another database. A person could scan through the database and visually check the values of each row or selected columns or selected rows. There are browsers to support this, but it is a very tedious and time consuming job, even with a good browser. In the case of very large databases, it is practically impossible. Therefore an automated data validation process is required (see Fig. 2: Data Validation Process)

4.1. Validating data content by means of a customized program

One alternative is to write a program which will read through the database, fetching the next record or selecting records according to some search criteria. Then the contents of the record are compared with the rules coded into the program or with the contents of another table used as a reference. This method is appropriate, but it requires time and effort to prepare. The program must be written by a programmer with knowledge of the database and how to access it. Not only that, but this solution may also be erroneous and has to be debugged.

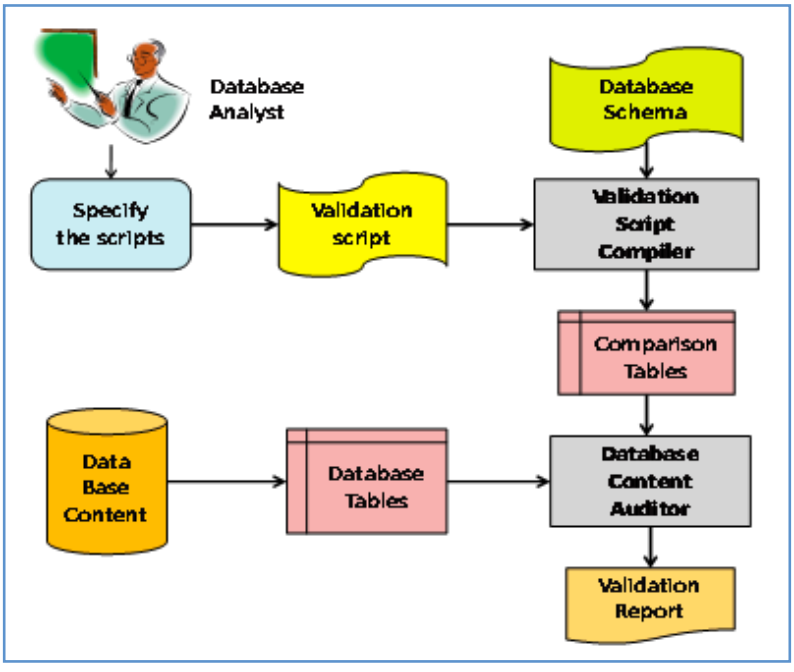


Figure 2: Data Validation Process

4.2. Validating data content with a test script

Another alternative is to have an interpretable specification – a script – which can easily be composed by someone familiar with the data. In the script, rules are formulated for validating the rules of selected attributes in selected records. This assumes, of course, that the auditor who writes the script knows what should be in the database. To support him, the auditor can use another database as a basis of comparison, but the auditor must establish the connections between two databases [27]. To write an interpretable specification, the auditor needs a language with which he/she can:

- specify which records with which keys should be in a table. The keys can be taken from the foreign keys of associated tables,
- specify which records with which keys should not be in a table. This can be achieved by comparing tables with one another,
- compare the content of a selected column in one table with the content of a corresponding column in another table,
- compare columns of the same table with each other,
- compare the values of a column with a given constant value,
- compare the values of a column with a range of values, i.e. boundary analysis,
- compare the values of a column with a set of expected values, i.e. equivalence class,
- compare the values of a column with the result of an arithmetic or logical expression,
- compare the values of a column with the concatenated values of other columns and constants [28].

All of the asserted comparisons listed above can be a subject to a logical expression, i.e. they will only be executed if that condition is fulfilled. The condition makes the values of one column dependent on the values of another column or on the primary key, e.g.

This_Column E RANGE (1:50) if (Mainkey > 1000);

There are many variations to the way these assertions can be formulated. The auditor can define discrete values as well as value ranges and equivalence classes, relations between data values and results of computations. The assertion language is based on the Schematron schema validation rules as defined in ISO/IEC Standard 19757 [30].

4.3. Executing the data validation

The data auditor formulates his assertions about the data in an assertion script for each database table and submits it to the compiler. The compiler checks the syntax and confirms the data names against the database schema. If errors are found, the auditor must correct the script. If the script is correct, an intermediate code is generated with the qualified data names and constant values referred to. Afterwards a validation run is started for each target table. The validation job processes the target table sequentially and checks one row after the other against the specified rules. If a row is missing or redundant, or if it contains invalid values, it is listed out by key in the validation report together with the expected values. With this report, the auditor can easily recognize incorrect content and prepare appropriate error reports. The data correctness ratio is the inverse relationship of the number of erroneous attributes relative to the total number of attributes in the database [31].

5. Summary

The two approaches to evaluating data quality have already been used in several measurement and testing projects with positive results. However, until now the two approaches have never been used together. The data assessment process proposes combining static schema analysis with data content validation [32]. First the rules have to be defined together with the users and the metrics weighted according to the user goals. Secondly, the database schemas will be statically analyzed and the structures measured. Thirdly, the validation scripts will be composed together with the customer based on what the customer believes the values of the columns should be. Finally, the database contents are examined and the exceptions documented. After that a report can be written on the state of the data quality.

The outcome of a data quality assessment depends to a great extent on the setting of the benchmark measurement values and the weighting of the metric values. These may be adjusted prior to running the measurement jobs. The user may attach more importance to data accessibility and less to data independence, or he may consider data flexibility to be more important than testability. Users may also be more or less tolerant of improved data contents. For one user it may be sufficient to have a correctness ratio of 95%, for another nothing less than 100% is acceptable.

These processes are based on practical experience from many projects of the last years. Because of the lack of tools for assessing the different aspects, a set of self-developed tools grows with these projects. E.g. the tools DLIAudit, ADAAudit and SQLAudit check the rules for IMS, ADABAS, DB-2, Oracle and MS-SQL databases. The result is a deficiency report of schema rule violations. SQLAudit also counts about 35 database attributes and computes 6 complexity and 6 quality metrics. All 12 metrics are ratio metrics derived from a relation of one set of attributes to another. The database metrics are collected in the form of a metrics report. In addition, they are exported as an XML file to be useable to anyone who wants to use them, for instance to store in a metric database or to further process. These metrics can create the basis for a benchmark, without which it is impossible to really evaluate anything. The tool Datatest offers a language based on predicate logic expressed in assertions with which the auditor can specify the data content. This was often used for validating the content.

On the bottom line, quality is relative. It is relative to the quality goals set. That is why the Goal-Question-Metric model established by Basili and Rombach long ago is still valid [33]. The user must first set goals, then he/she must ask how to judge their fulfillment and thirdly he/she can define metrics to measure the degree of their fulfillment. This method applies equally well to the assessment of data quality. The data analyst must begin by setting goals, goals such as portability, extendibility and flexibility. Then metrics must defined to measure the degree to which they are fulfilled by the current database. Finally, they must be measured and evaluated. Without clearly defined, measurable goals, it is not possible to judge anything.

6. References

[1] Belady, L., Lehman, M.: "A Model of Large Program Development", IBM Systems Journal, Vol. 15, Nr. 3, 1976

[2] Blaha, M./ Premerlani, W.: "Observed Idiosyncracies of relational Database Design", IEEE Proc. of 2nd WCRE, Toronto, July, 1995, p. 116

[3] Premerleni, W./ Blaha, M.: "An Approach for Reengineering of relational databases", Comm. Of ACM, Vol. 37, No. 5, May, 1994, p. 42

[4] Tayi, G.-K./Ballou, D.: "Examining Data Quality", Comm. Of ACM, Vol. 41, No. 2, Feb. 1998

[5] Date, C.J.: An Introduction to Database Systems, Addison-Wesley Pub. , Reading Mass., 1975

[6] Redman, T.C.: Data Quality for the Information Age, Artech House, Boston, 1996

[7] CW: Computer Weekly, Nr. 26, June 1998, p. 1

[8] Yong, J.K./Kishore, R./Sanders, G.L.: "From DQ to EQ – Understanding Data Quality in the Context of E-Business Systems", Comm. Of ACM, Vol. 48, No. 10, Oct. 2005, p. 75

[9] Blaha, M.: "A copper Bullet for Software Quality Improvement", IEEE Computer, Feb. 2004, p. 21

[10] Wand, Y./Wang, R.: "Anchoring Data Quality Dimensions in Ontological Foundations", Comm. Of ACM, Vol. 39, No. 11, Nov. 1996, p. 86

[11] Kaplan, D./Krishnan, R./Padman, R./Peters, J.: "Assessing Data Quality in Accounting Information Systems" in Comm. of ACM, Vol. 41, No. 2, Feb. 1998

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

[13] [13] Kan, S.H.: Metrics and Models in Software Quality Engineering, Addison-Wesley, Boston, 2001

[14] International Function-Point User Group - IFPUG: Counting Practices Manual, Release 4.1. IFPUG, Westerville, Ohio, 1999

[15] Sneed, H.: Die Data-Point Methode, Online, Zeitschrift für DV, Nr. 5, May 1990, S. 48

[16] Card, D., Agresti, W.: "Measuring Software Design Complexity", Journal of Systems & Software, Vol. 8, 1988, S. 185

[17] Hainaut, J-L.: "Strategies for Data Reengineering", IEEE Proc. of 9th WCRE, Richmond, Oct. 2002, p. 211

[18] Aiken, P.: Data Reverse Engineering, McGraw Hill, New York, 1996

[19] Blaha, M.: A Manager's Guide to Database Technology – building and purchasing better Applications, Prentice-Hall, Englewood Cliffs, 2001

[20] Orr, K.: "Data Quality and System Theory" in Comm. of ACM, Vol. 41, No. 2, Feb. 1998

[21] Jackson, M.: Principles of Program Design, Academic Press, London, 1975

[22] Aiken, P./Muntz,A./Richards,R: "DOD Legacy Systems Reverse Engineering Data Requirements", Comm of ACM, Vol. 37, No. 5, Feb. 1994, p. 26

[23] Brathwaite, K.: Systems Design in a Database Environment, McGraw-Hill, New York, 1989, p. 106

[24] Wang, R.: "Total Data Quality Management", in Comm. of ACM, Vol. 41, No. 2, Feb. 1998

[25] ISO/IEC: "Software Product Evaluation - Quality Characteristics and Guidelines for their Use" ISO/IEC Standard ISO-9126, Geneva, 1994

[26] Howden, W.: "The Theory and Practice of Functional Testing", IEEE Software, Vol. 2, No. 5, Sept. 1985, p. 6

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

[28] Sneed, H.: "Bridging the Concept to Implementation Gap in System Testing", IEEE Proc. of TAICPART Workshop, Windsor, G.B., Sept. 2009, p. 172

[29] Sneed, H.: „Reverse Engineering of Test Cases for Selective Regression Testing", Proc. of European Conference on Software Maintenance and Reengineering, CSMR-2004, IEEE Computer Society Press, Tampere, Finnland, March 2004, S. 69

[30] ISO/IEC: "Rule-based Validation – Schematron", ISO-IEC Standard 19757, International Standard Organization, Zürich, 2006

[31] Sneed, H.: "Testing a Datawarehouse – an industrial challenge", IEEE Proc. of TAICPART Workshop, Windsor, G.B., August, 2006, p. 203

[32] Sneed, H./Baumgartner,M./Seidl,R.: Software in Zahlen, Hanser Verlag, München/Wien, 2010, S. 197

[33] Basili, V., Caldiera, C., Rombach, H-D.: "Goal Question Metric Paradigm", Encyclopedia of Software Engineering, Vol 1, John Wiley & Sons, New York, 1994, S. 5

> biography

Richard Seidl

Since early 2011 Richard Seidl has been test expert and test manager at GETEMED Medizin- und Informationstechnik AG, Teltow (Germany). His work focuses on the planning and implementation of software and hardware tests of medical devices. After graduating in communications engineering, Richard Seidl began working as software engineer and tester in the of banking field. From 2005 to 2010 Richard Seidl was test expert and test manager at ANECON GmbH, Vienna (Austria). In 2006 he acquired the Full Advanced Level Certification of the ISTQB®. In the following year he qualified as IREB Certified Professional for Requirements Engineering and as Quality Assurance Management Professional (QAMP). He published the books „Der Systemtest" and „Software in Zahlen" together with Harry M. Sneed and Manfred Baumgartner.

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 as a test project leader. Now, at the end of his long career, he has returned to the role of a software tester for the ANECON GmbH in Vienna. Parallel to his project work Harry Sneed has written over 200 technical articles and authored 18 books including four 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.

Beware!.... Model-based testing

by Erik van Veenendaal

What it should not be!

Having been in the software testing profession for over 20 years, reading or hearing about model-based testing has always given me a sense of warning. Too often I have been disappointed over the years, great papers and enthusiastic test professionals (and experts) presenting the latest about model-based testing..... Often presented as a silver bullet, no more test design techniques and activities, test cases would be generated and of course automated. Great savings and benefits promised, the world would change and traditional methods were no longer needed. Practice was always very different from theory, and I hardly ever saw their ideas being applied in projects. It stems from these experiences in the recent and far past, that I'm always very cautious when I hear people presenting ideas for model-based testing.

A definition in line with the above:

> ***"Model-based testing (MBT) is the automatic generation of software test procedures, using models of system requirements and behavior. Although this type of testing requires some more up-front effort in building the model, it offers substantial advantages over traditional software testing methods."***

Of course, the definition comes with all these great advantages that will be achieved:

- Project maintenance is lower. You do not need to write new tests for each new feature. Once you have a model, it is easier to generate and re-generate test cases than it is with hand-designed test cases.
- Design is fluid. When a new feature is added, a new test action is added to run in combination with existing test actions. A simple change can automatically ripple through the entire suite of test cases.
- High coverage. Tests continue to find bugs, not just regressions due to changes in the code path or dependencies.
- Model authoring is independent of implementation and actual testing, so that these activities can be carried out by different members of a team concurrently.

You may wonder why we are not all doing this. Just apply MBT and all testing problems will go away.

What it should be!

I hope I did not offend too many test professionals with the first section. I hope, however, that I have alerted most of you. Of course, model-based testing is a great idea, but it is not a silver bullet (as nothing ever is) and some discussion is needed to put things into perspective.

A much better definition can be found on Wikipedia:

> ***"Model-based testing is the application of model-based design for designing and optionally executing the necessary artifacts to perform software testing. Models are used to represent the desired behavior of the System Under Test (SUT)."***

Model-based testing ≠Test automation

Note that in the second definition the word automation is not present. This puts things into perspective. Model-based is all about using (formal) models, e.g., UML based, to design test cases. The model is a tool to better understand the SUT, and will be used as a starting point to design test cases. Of course, some formal models allow for automation and generation of test cases, but this is not a mandatory part of model-based testing. And, as we all know, automation is never as easy as it looks in demos.

All testing is model-based

If one takes a step back, isn't every requirements specification (whether formatted using formal models or just plain text) a model of the desired behavior of the SUT? When designing test cases, we use the requirement to establish our test conditions and test cases. So in fact requirements-based testing (as explained in many traditional methods) is also model-based testing. This puts things into perspective.

Large up-front investment

Model-based testing requires a substantial larger up-front effort. Of course, we all know up-front investments will pay off later, but somehow this is always very difficult to sell to management. It is easy to say, but usually very difficult to implement. A parallel can be made with reviews. We all know they pay off later, but recent

studies show that after all these years only 25% of organizations practice reviews in a professional way. This puts things into perspective.

Exploratory testing is not covered

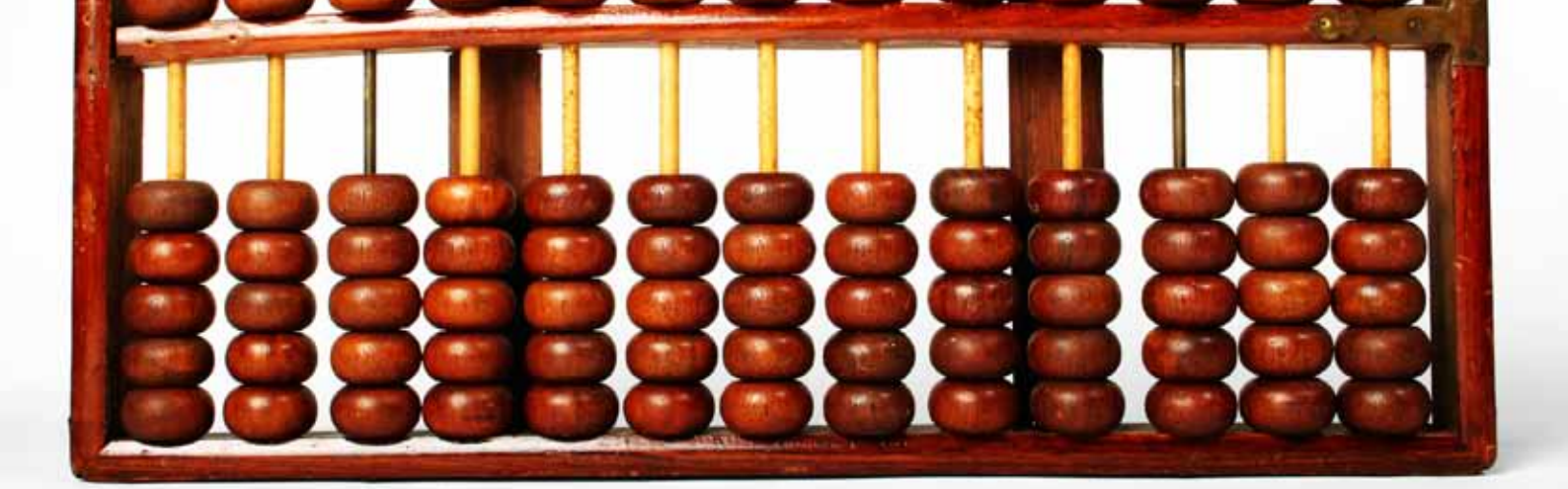
Many model-testing "experts'" claim it replaces all other testing. I believe that over the last decade we have learned that experienced-based testing, e.g., exploratory testing, is a very powerful tool to find defects. In some projects it is "just" an add-on to more formal test design techniques, and in some projects it is even the only testing strategy being applied. Model-based testing and exploratory testing can be combined, but come from a very different way of thinking. It's not all about processes, people are a key to success as well.

There are of course many more issues to be discussed in the context of model-based testing, each with its advantages and disadvantages. I'm sure you will find many of these in this issue of Testing Experience.

Good luck with your model-based testing, but beware.....

> biography

Erik van Veenendaal
(www.erikvanveenendaal.nl) is a leading international consultant and trainer, and a widely recognized expert in the area of software testing and quality management with over 20 years of practical testing experiences. He is the founder of Improve Quality Services BV (www.improveqs.nl). 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 various domains for more than 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, vice-president of the International Software Testing Qualifications Board (2005–2009) and currently vice chair of the TMMi Foundation.



The advantages of model-based testing for financial services firms

by Govind Muthukrishnan

The success or failure of today's banks, insurance companies and capital market firms is intricately linked to information and system quality. IT quality is achieved and maintained through software testing, and one of the most time-consuming and resource-intensive tasks of software testing is the creation of test suites. Post-crisis, financial institutions have been investing in significant IT initiatives to meet new regulatory requirements, address growing mobile and internet channels, and differentiate themselves from the competition. Model-Based Testing (MBT) can help financial firms develop systems and applications that better meet user requirements without adding significant cost.

Under current testing methods, test professionals must manually create test scripts and use cases. Since financial systems are often complex and customized, creating test assets requires both testing skills and domain expertise in banking, capital markets or insurance. By using MBT tools, financial institutions can create a model to describe the application, system or object under test and automatically generate hundreds of test cases from the one model. Since models and model components are more flexible and robust than manually created test assets, a single model can generate multiple test cases in far less time than it would take to write the same number of cases manually. Models can also be reused to test other applications with similar functionality.

We see a lot of similarities between the automation wave in the testing industry about 10 years ago and the MBT wave that is emerging today. Every testing service provider has recommended MBT to make sure they are in the race. The challenges of MBT are not that different from the challenges we faced in the automation era. Before implementing MBT, you should:

- Carefully identify the areas of applicability, methods used in building the models, and choice of tool sets;
- Calculate the ROI before making huge investments; and
- Obtain commitment to the approach over the long-term, not just for quick wins.

Model-based testing:

- Significantly reduces the time required to complete functional testing, bringing measurable cost reductions; and
- Provides increased test coverage, allowing test professionals to identify more defects.

With experienced and domain knowledgeable test professionals, the time required to thoroughly test a newly implemented, core financial industry application system such as insurance claims processing or commercial lending can be reduced by as much as 30% through model-based testing.[1]

Why use models?

For test professionals, model-based testing is a compelling methodology because models can be used to:

- Precisely define functional requirements to better match user expectations;
- Reduce planning time by providing components of functional specification documents; and
- Automate test case generation, creating of more test cases in less time.

The goal of model-based testing is to create a generic set of model components that can be reused in every new project. Another approach, which is particularly helpful in a complex, financial-enterprise-IT environment, is to build model components specific to an application system which can be reused when incorporating business-process-improvement flows.

Model-based testing also lets you adapt models quickly for similar systems. For example, Capgemini created models to test a commercial lending system which was being launched in Bulgaria. The same models were quickly adapted to test the same application during the rollout to Albania and Romania.

Recently, papers have been published about the limitations of MBT. At Capgemini, we have been using MBT in carefully chosen areas with great success. For example, MBT is a good solution to test package-based applications and stable release-based applications. Model-based testing is not a good fit in the typical application development phase where the design is changing. Our strategy has been to identify the areas of application carefully instead of jumping in with MBT at the first available opportunity. This helps our clients have more realistic expectations for MBT initiatives.

1 Based on Capgemini experience using model-based testing on testing engagements for financial services institutions, compared to previous manual testing method in use.

Building a model: Key considerations

- **Identify the best source for requirements**. Each system under test can be unique to the business process that it will support. So identifying the best approach to build the models for model-based testing can be challenging. Functional requirements – like those derived from use cases – can serve as a starting point for the creation of the model.
- **Select the right model**. Most MBT tools support the development of models using unified modeling language (UML). When building models, test professionals can use the software-application-development model or create a separate test model. Both have their advantages; using a development model is faster , but using a test model uniquely developed for the specific project is more targeted and does not replicate errors in the development model
- **Consider structured analysis techniques**. By building a model of a system in a way that supports structured analysis, the desired behavior of the system can be tracked. For the testing team, a structured approach is very important since test case scenarios are comprised of sequences of interactions with the system; each scenario having a pre-defined "correct" processing outcome. The goal of functional testing is to exercise all possible combinations of user-system interactions. Capgemini uses a proprietary testing management methodology called TMap® to ensure structured test design techniques are incorporated in our testing projects.
- **Design the model**. We use the TMap® test design technique checklist to quickly create models using pre-packaged components. Steps include preparing a set of generalized model components; integrating or reusing generalized components when building models for new requirements; and identifying common operations for systems under test which can form a set of model components specific to the product. By applying these testing techniques, test professionals can develop a set of generic packages to test standard functions such as Windows compliance, screen validations, specific field tests or business rules. These generic packages can be customized for the products or applications in use based on company-specific requirements.

The business advantages of model-based testing

Model-based testing brings significant advantages:

- **Supports reuse**. Reusable repository of model and model components provides proven productivity gains and time savings.
- **Decreases rework and errors**. By eliminating ambiguous, unclear, and incomplete requirements through up-front definition and validation, basic flows can be verified and errors minimized.
- **Provides structured, traceable change management**. Model-based testing makes it easier to manage changes in the project by merely updating the model.
- **Increases test case effectiveness**. Test cases generated by model-based testing techniques such as Capgemini's proven TMap® approach are more effective at providing optimum coverage and quality.
- **Reduces total test preparation time**. Reusable models and model components help minimize test preparation time by allowing hundreds of test cases to be created with one click.
- **Ensures full coverage for validation**. Checklists assure all validation aspects. For each field, a model component checks all possible conditions across all test cases so there is no chance of missing a validation check while building a new model. For example, once a model component is built to validate credit card numbers, every test case created with the model will have that validation check built in.
- **Lowers efforts for integration testing**. Since it's easier to integrate functional or software components into the model, integration testing can be accelerated.

Conclusion

Model-based testing, a methodology developed to automate the functional-testing phase of a system implementation project, can generate significant software quality assurance benefits. In addition, it can allow financial services institutions to reduce testing time by as much as 30% over traditional testing methods.

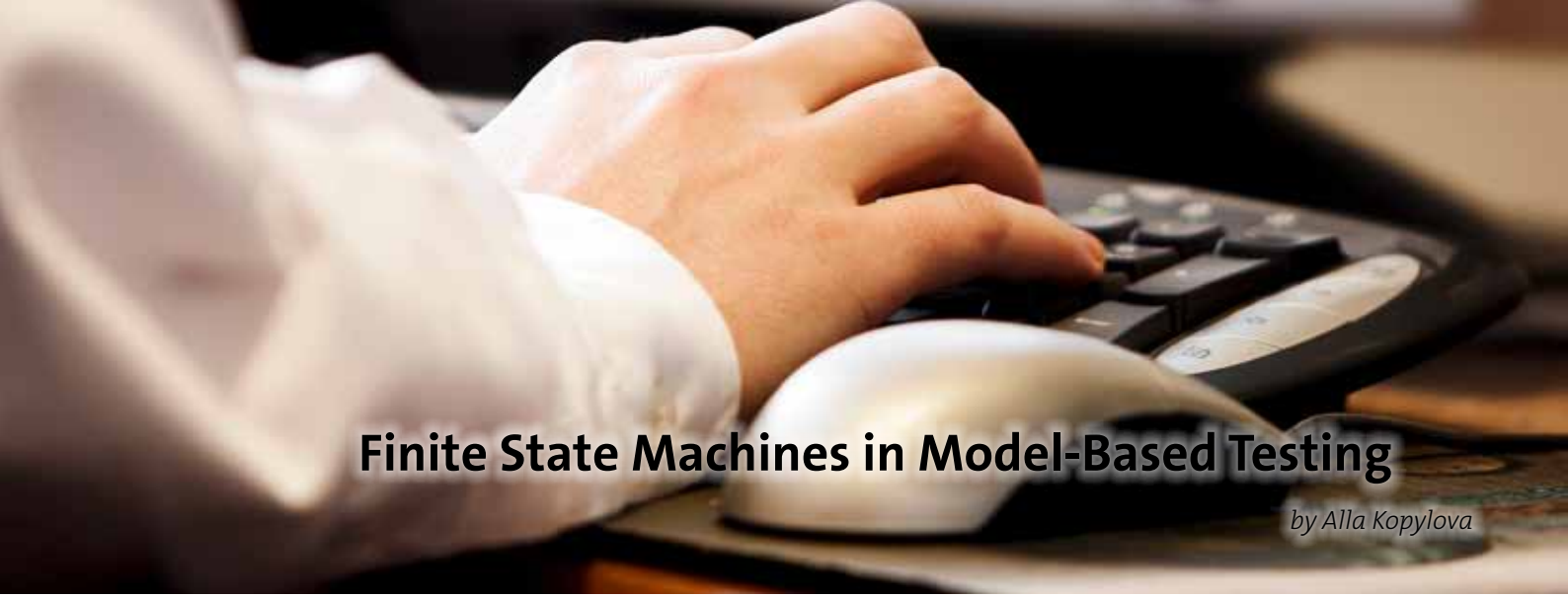
By creating models specific to IT or business-application environments, test professionals can clarify business requirement definitions and automate the generation of test cases to thoroughly check system functionality and identify system errors that may frustrate all users – internal stakeholders and financial services customers alike. Once created, the model-based testing application model can be reused to support future testing of the target applications, including changes to support business process improvements or new systems software releases. For financial services institutions, the use of model-based testing can simplify the testing of complex, core financial systems by building a library of models and model components that can be reused across different divisions or regions that have customized versions of the same system.

> biography

Govindarajan Muthukrishnan *is Vice President, Global & Rightshore Testing Leader, Financial Services Global Business Unit which includes over 3,800 test professionals. Govind is responsible for managing the global testing practice devoted to financial services, increasing operating efficiency and creating differentiators through various technology and domain focused Centers of Excellence and Innovation Labs. He is a testing professional with over 16 years of experience in working with several testing tools and technologies.*

Prior to joining Capgemini as part of the Kanbay acquisition in 2007, Govind led testing teams serving one of the world's largest banks. He also ran Kanbay's technology services, which included eSolutions, business intelligence, infrastructure management, testing and PeopleSoft. Govind has been a pioneer in bringing testing best practices to the industry and participated in industry conferences in India and North America such as QAI, SPIN and STAREAST.

A management graduate from the Indian Institute of Management, Govind began his testing career at Ramco Systems and Polaris Software Labs before moving over to Cognizant where he was instrumental in building a testing practice from zero to over 2,000 professionals over a period of five years.



Finite State Machines in Model-Based Testing

by Alla Kopylova

One of the desirable things in software development is to have something that defines the system's behavior in a concise way. It is hard to imagine that systems could be developed without having been previously analyzed and modeled. When the time comes to test software, we also need an oracle that can be used for development of the test cases. In this connection, developing behavior models comes first.

To build a model means to form a mental representation of the system's functionality that can be used during the system implementation and for further testing purposes. What does 'model' mean to us? Possible answers could be:

- A model is used to develop an understanding of the system. It gives a complete description of software behavior.
- A model is used to give the complete understanding of the target users and their actions.
- A model is used to provide examples of the input and output of each software user.
- A model presents the input and output logic of the system and dataflow through the application. That's why it is considered good practice to create models before the development process is started.
- Behavior models are also a source for test suite creation, so they are widely used in Black Box testing.

How could models be used in testing?

A model represents the desired behavior of the system under test and also gives hints for the test strategies that could be chosen as well as the testing environment.

A model that describes software is usually an abstract presentation of the desired behavior. Here the domain analysis that is performed to define the domain, collect information about it and produce a domain model comes in handy. The possible sources for the domain analysis are:

- Existing systems with design and requirements documentation;
- Standards;
- Customer ideas of how the system should behave (for this, the MoSCoW prioritization method could be used to prioritize customer desires regarding system functionality).

The results of the domain analysis are used to derive high-level functional test cases. These are rather abstract test cases that could not be executed against software under test because they are at the wrong level of abstraction.

The next thing that should be done is to develop the executable test suite based on the high-level test cases.

The algorithm of model-based testing is quite simple:

- Build the system model;
- Generate expected inputs;
- Generate expected outputs;
- Run tests over the system;
- Compare actual and expected outputs;
- Decide on further actions: whether to modify the model or generate more tests or stop testing.

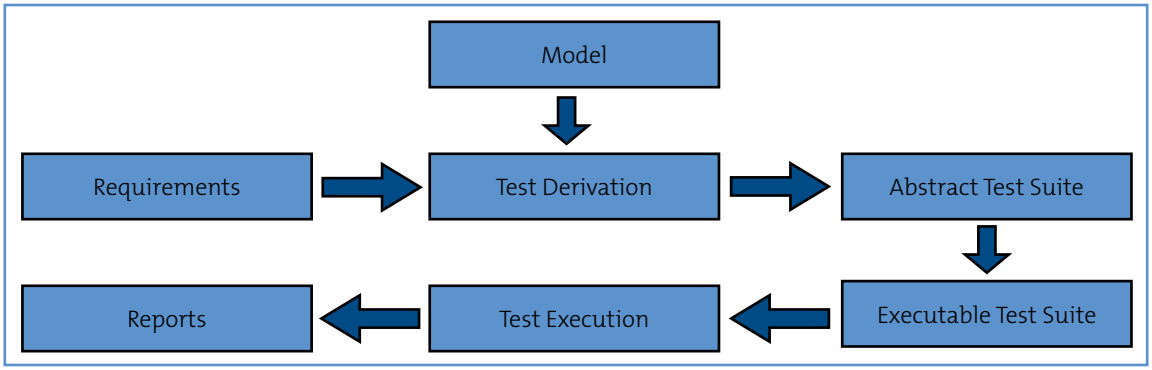


Fig. 1. Model-Based Testing Workflow

As for the test engineers, the most interesting part in this algorithm is test case development (generation of inputs and expected outputs). Model based testing has a set of approaches for developing functional test cases:

Test cases generation by theorem proving

The system is modeled as a set of logical expressions specifying the system's behavior. This model is partitioned into equivalence classes, where each class represents certain system behavior and can serve as a test case.

Test case generation by constraint logic programming

The system is described by means of constraints. Solving the set of constraints can be done using Boolean solvers and the solutions found can serve as test cases.

Test case generation by model checking

A model checker detects paths (where property of a requirement is satisfied) and counter-examples (where path is violated). These paths are test cases.

Text case generation using event-flow model

An event-flow model represents all possible sequences of events that can be executed on the GUI for end-to-end testing.

Test Case generation by using the Markov chains model

The Markov chains model is an efficient way to handle model-based testing. The particular aspect about Markov Chains is that the next state depends only on the current state and is not referenced to past states and events. Two artifacts construct Markov chains:

- Finite state machines, representing all possible usage scenarios of the system (this helps to know what can be tested);
- Operational profiles, which qualify the finite state machine to represent how the system is or will be used statistically (helps to derive operational test cases)

Let's dwell on the technology of using finite state machines for behavior modeling.

Each software program has a determined number of states in which it can be found at the current moment. These states are defined with the values of the system variables and their combinations.

The bug hypothesis is the following: From the great variety of possible states in which system can be found, several were not implemented during development. In cases where such unknown combinations of input conditions occur during usage of the system, the software switches to the unknown state in which the system behavior differs from the expected behavior. This can result in failures ranging from minor functionality inconsistences up to critical issues in system behavior.

The finite state machine model is a fundamental testing model that could be applied to a great variety of object-oriented software applications. It is likely to be used for testing the system logic, and it is the most efficient way to check applications that are controlled via menus.

To build a feature state machine for some part of an application, the following steps should be followed:

1. In the domain analysis phase, all possible system inputs, outputs and states should be defined. All states should be properly checked: missing states should not be present, neither should excessive states (excessive states mean that the system has an unknown behavior).

2. All possible transitions between states should be defined. The software should be designed in a way where it cannot get into the 'wrong' state. Each transition should have an input and an output.

After the model for the system has been developed, the following documents are generated:

- Definition of the system behavior (list of all systems functions);
- Scheme that connects state machine with its environment (interfaces);
- Transitions graph (defining the state machine behavior).

After the test cases for the developed model have been generated, the additional artifacts are available:

- Test cases with expected outputs;
- Test results from which the reliability of the software can be estimated.

Model based testing is an efficient testing technology that can be applied for different software systems, such as menu-driven software, protocols, device drivers, installation software, etc. Software has a finite number of states or combinations of parameters that describe the software at the current moment. This knowledge is used to describe the software behavior and can be used for test cases creation. The finite state machine technique provides the most efficient way to test software logic, but the main drawback of this approach is that it is hard to use for systems with a large number of states and transitions between them.

> biography

Alla Kopylova
has four years of experience in the IT sector as Quality Assurance engineer. Having started her career in testing of medical software, she then moved to the mobile applications sector where she holds the roles of Senior QA Engineer and Business Analyst at Quickoffice Inc. She holds a Master Degree in Computer Science and is an ISTQB® certified test professional.

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

Picture Credits

Index of Advertisers

Knowledge Transfer – The Trainer Excellence Guild

Díaz Hilterscheid

Specification by Example: from user stories to acceptance tests
by Gojko Adzic

The winner of the MIATPP award presents: This two-day workshop teaches you how to apply emerging practices for managing requirements, specifications and tests in agile and lean processes 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.

The workshop is aimed at testers, business analysts and developers and based on Gojko Adzic's books Specification by Example and Bridging the Communication Gap. Through facilitated exercises and discussion, you will learn:

- the key principles of specification by example, behaviour-driven development, effect mapping
- how to create and maintain effective specifications and tests in agile and lean projects
- how to ensure a shared understanding of testers, developers, business analysts and stakeholders on future functionality
- how to extend specifications with examples to create a single source of truth for testing and development
- how to avoid functional gaps and inconsistencies in specifications and tests
- how to run specification workshops to facilitate collaboration
- best practices for designing specifications and automating them as tests
- how to create a living documentation system to facilitate change and improve your process long-term
- how other teams, from small web startups to large distributed teams in investment banks, apply specification by example in their contexts
- how to apply this process in your project

Date	Place	Price
March 22–23, 2012	Berlin	€ 1,200 + VAT

TDD .NET
by Lior Friedman

This three-day program provides a highly interactive exploration of unit test automation, principles and practices. Companies such as Microsoft, Google, and IBM have already realized the potential that lies in Unit Test automation. Following these practices reduces the amount of defects in your software decreasing the effort involved in product development and result in more satisfied customers. Learn how to use frameworks for test writing and how to isolate your code in an efficient and easy manner.

Objectives:
- Writing basic unit tests using MsTest/NUnit
- Learning principles of Test Driven Development
- Experience Best Practices of unit testing
- Understanding the difference between unit tests and acceptance tests.
- Learn how to test complex object hierarchies
- Understand the difference between Interaction Testing and state based testing.
- Understand the principles of Isolation using modern isolation frameworks.
- Learn advanced Mocking techniques
- Learn how to leverage frameworks to ease authoring of tests
- Real Life examples.

Date	Place	Price
May 07–09, 2012	Nürnberg	€ 1,450 + VAT

Building Better Software With Business Stories
by Paul Gerrard & Susan Windsor

It's all about communication, communication, communication…..
Using business stories and examples to build better software is an approach that is currently being adopted by forward thinking organisations and individuals. Place business stories at the heart of software development, regardless of your development approach.

As an individual you can attend one of our public courses (2012 dates for London given below), or we can run this in-house for your team. Either way you will benefit from learning about some new and valuable techniques, and you'll receive a copy of our latest Business Story Pocket Book for future reference.

In summary, the key benefit of this course enables all project participants to communicate in the natural language of Business Stories:

- Business analysts derive Business Stories from requirements to confirm with stakeholders that requirements can be trusted.
- Developers take the same stories and use them to help design their code and automatically test it
- Test analysts take the same stories and add detail required for system and acceptance testing as software becomes available.

Date	Place	Price
June 21–22, 2012	Berlin	€ 1,250 + VAT

Website:
http://www.diazhilterscheid.de/en/knowledge_transfer.php

Training with a View

Díaz Hilterscheid

Dates *	Course	Language	Place	Price
05.04.12–05.04.12	Anforderungsmanagement	German	Berlin	499,-
23.04.12–27.04.12	CAT – Certified Agile Tester (DE)	German	Berlin	2100,-
19.03.12–23.03.12	CAT – Certified Agile Tester (EN)	English	Amsterdam	2100,-
10.04.12–11.04.12	HP Quality Center	German	Berlin	998,-
14.05.12–15.05.12	HP QuickTest Professional	German	Berlin	998,-
27.03.12–29.03.12	IREB® Certified Professional for Requirements Engineering – Foundation Level (DE)	German	Berlin	1400,-
06.06.12–08.06.12	IREB® Certified Professional for Requirements Engineering – Foundation Level (DE)	German	Berlin	1400,-
16.04.12–18.04.12	IREB® Certified Professional for Requirements Engineering – Foundation Level (EN)	English	Helsinki	1600,-
05.03.12–08.03.12	ISTQB® Certified Tester Foundation Level	German	Berlin	1800,-
02.04.12–04.04.12	ISTQB® Certified Tester Foundation Level – Kompaktkurs	German	Stuttgart	1495,-
16.04.12–18.04.12	ISTQB® Certified Tester Foundation Level – Kompaktkurs	German	Frankfurt	1495,-
02.05.12–04.05.12	ISTQB® Certified Tester Foundation Level – Kompaktkurs	German	Berlin	1495,-
29.05.12–31.05.12	ISTQB® Certified Tester Foundation Level – Kompaktkurs	German	Nürnberg	1495,-
12.03.12–16.03.12	ISTQB® Certified Tester Advanced Level – Technical Test Analyst	German	Berlin	2100,-
07.05.12–11.05.12	ISTQB® Certified Tester Advanced Level – Technical Test Analyst	German	Mödling/Austria	2100,-
26.03.12–30.03.12	ISTQB® Certified Tester Advanced Level – Test Analyst	German	Frankfurt	2100,-
26.03.12–30.03.12	ISTQB® Certified Tester Advanced Level – Test Analyst	German	Mödling/Austria	2100,-
26.03.12–30.03.12	ISTQB® Certified Tester Advanced Level – Test Analyst	German	Stuttgart	2100,-
07.05.12–11.05.12	ISTQB® Certified Tester Advanced Level – Test Analyst	German	Köln	2100,-
19.03.12–23.03.12	ISTQB® Certified Tester Advanced Level – Test Manager	German	Köln	2100,-
16.04.12–20.04.12	ISTQB® Certified Tester Advanced Level – Test Manager	German	Stuttgart	2100,-
05.03.12–09.03.12	ISTQB® Certified Tester Advanced Level – Test Manager	German	Mödling/Austria	2100,-
16.04.12–20.04.12	ISTQB® Certified Tester Advanced Level – Test Manager	German	Stuttgart	2100,-
21.05.12–25.05.12	ISTQB® Certified Tester Advanced Level – Test Manager	German	Berlin	2100,-
12.04.12–13.04.12	Testen für Entwickler	German	Berlin	800,-
26.03.12–27.03.12	Testmetriken im Testmanagement	German	Berlin	998,-

*** subject to modifications**
all prices plus VAT

IREB International Requirements Engineering Board

CAT Certified Agile Tester®

ISTQB® Accredited Training Provider

Kurfürstendamm, Berlin © Katrin Schülke

more dates and onsite training worldwide in German, English, Spanish, French on our website:
http://training.diazhilterscheid.com