December, 2008

printed in Germany

testing experience

The Magazine for Professional Testers

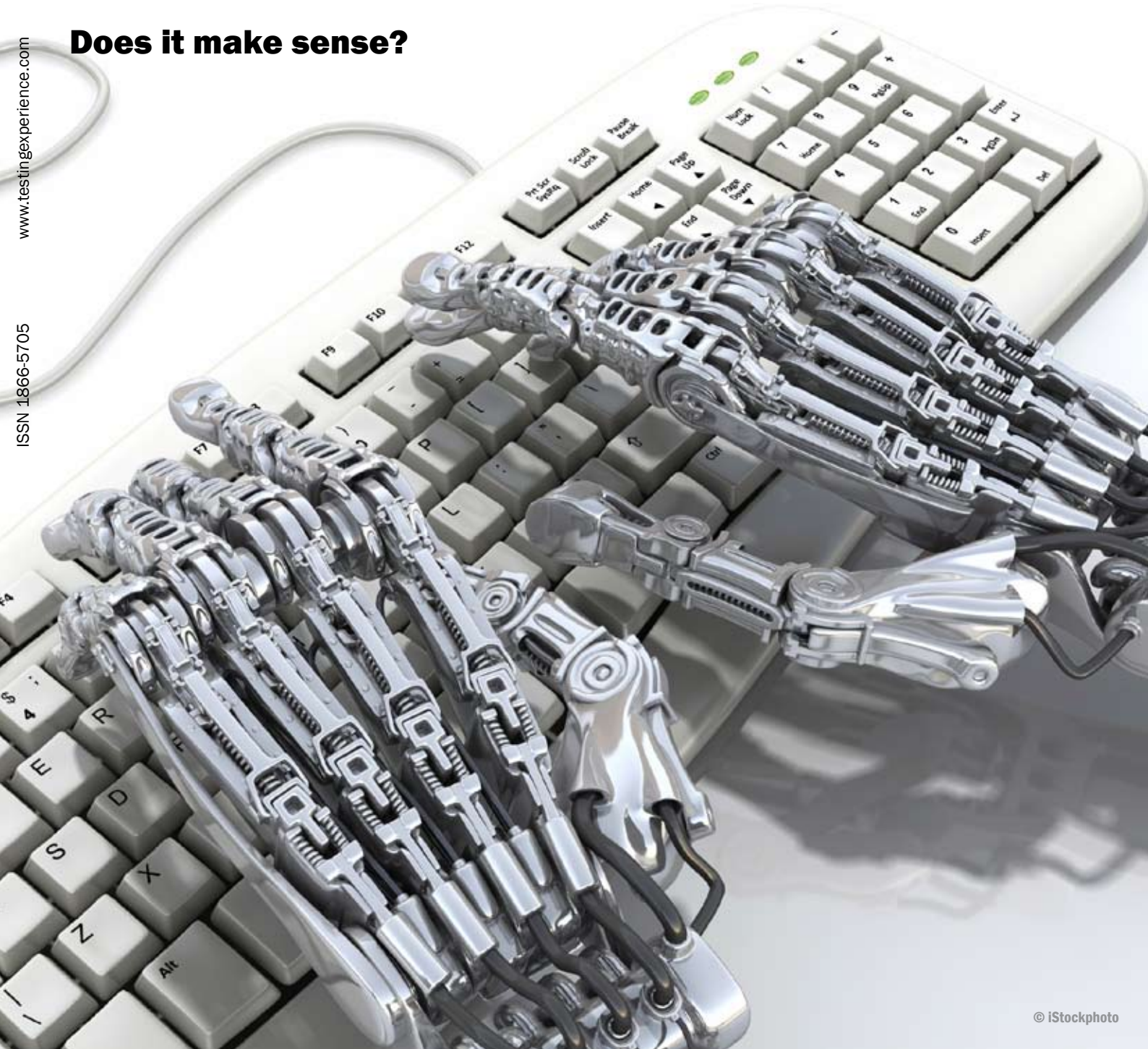
free exemplar

www.testingexperience.com

ISSN 1866-5705

Test Automation -

Does it make sense?



www.testingfinance.com

Testing & Finance 2009

The Conference for Testing Professionals in Financial Services

Call for Papers

June 22-23, 2009
Frankfurt am Main, Germany

Send your proposals till the end of January, 2009
to info@testingfinance.com

Exhibitors

Saxonia Systems
A vision of technology.

Compuware

hp

SELA GROUP

iSQI
International Software Quality Institute

PureTesting
Testing Thought Leadership

isacon

CaseMaker

ib-bank-systems

Supporting Organisations

VOB SERVICE

dpunkt.verlag

Díaz Hilterscheid

te testing experience

Dear readers,

"The one duty we owe to history is to rewrite it" said Oscar Wilde a long time ago. I don't know if we have to rewrite the history of test automation, but we do need to talk about the past, the present and the expectations for the next generation of tools and techniques.

Many of us have seen that test automation did not always work out and bring the expected ROI. We have met a lot of sales guys offering gold, but after scratching on the surface we saw that there was a lot of lead inside. There is no universal test automation tool for all your needs. You have to look for the best solution to suit your specific needs. Sometimes you can buy, sometimes you can use freeware and sometimes you have to build your own. However, we have also seen that there are a lot of cases where test automation makes sense and brings the expected ROI. It is certainly a matter of experience.

I think that the articles in this issue will give you a flavour about the test automation world outside and give you some hints for your daily work. I would recommend that take a look at Bj Rollinson's blogs. He has a lot of good hints and tools for test automation in it. Sorry Bj! You will have 150,000 hits in a few days.

The whole world talks and worries about the financial crash. In my opinion, it provides a good chance for the testing world and for the profession. The Germans said: where there is shadow, there is also light. Industry, financial institutions and governments have to spend money to ensure that their requirements are achieved. Test professionals have to focus more than ever on results. They have neither money nor time to lose. Techniques and the experience, based on a proper knowledge of the customers' requirements, are essential to complete within planned time and budget and to survive in the market. No more games! Just quality.

It is no coincidence that ISTQB Certified Tester Certifications have reached over 100,000 Certified Testers worldwide. I think this reflects the market needs and the objectives for a common knowledge and high levels of professionalism. The certification will, of course, not make you a tester within three or four days. Only experience does that. But you get the background!

We now have the fourth issue of Testing Experience and are very happy about the audience's acceptance. We had over 150,000 downloads for the third issue. Thanks!

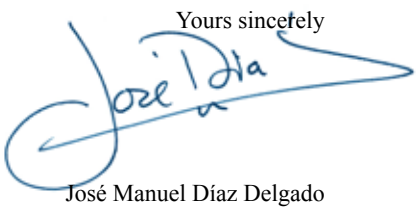
Thank you very much also for your compassion for not having a holiday during the summer time. I have to tell you that I did have a wonderful time in October in Delhi and Agra visiting also the "Taj Mahal". Wow! I belong to the those who have seen it. I spent some days at the Test2008 conference. I can only recommend you to visit it next year. It was a great conference. For Europeans, India is a country rich in contrasts, full of feelings and temptations. I was delighted with it and also with the people that I met there. An experience for life. I'm looking forward to going back soon.

I was also at the SIGIST in Israel. Go for the next one! It was a wonderful conference. Israel is an amazing country also for holidays with the family. I was with my family also in Bethlehem - West Banks, Palestinian terretories -. Only good impressions and safe!

Over Christmas, I will be four weeks in Gran Canaria visiting my family and enjoying the beach, the people and the sales! Should you also be there, drop me an email. We can go for a beer or two! I'm aware of the fact that I won't get any compassion after this.

Last but not least I would like you to note the dates for the next "Testing & Finance", which is going to take place in June 22-23, 2009 in Frankfurt, Germany. We expect over 500 people who are involved in Testing and Finance. See you there.

Yours sincerely

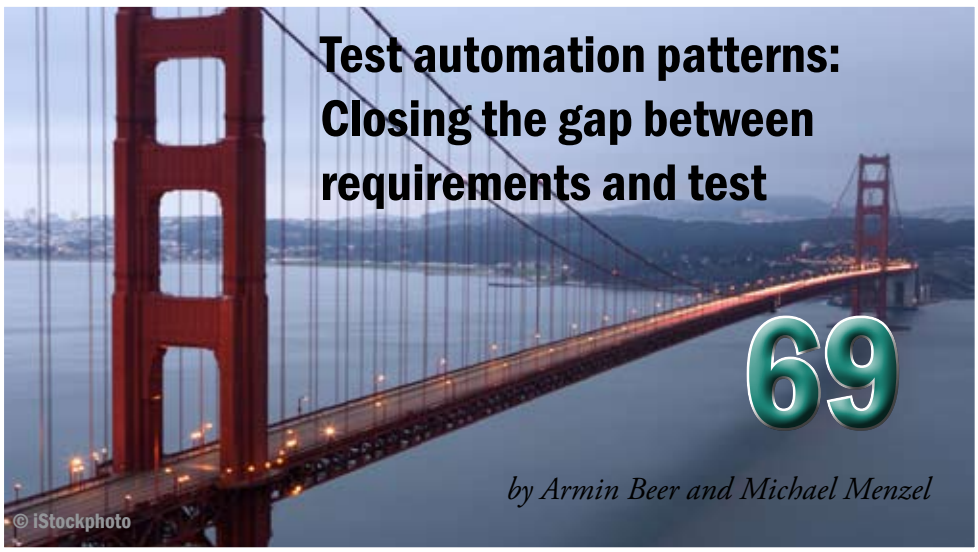


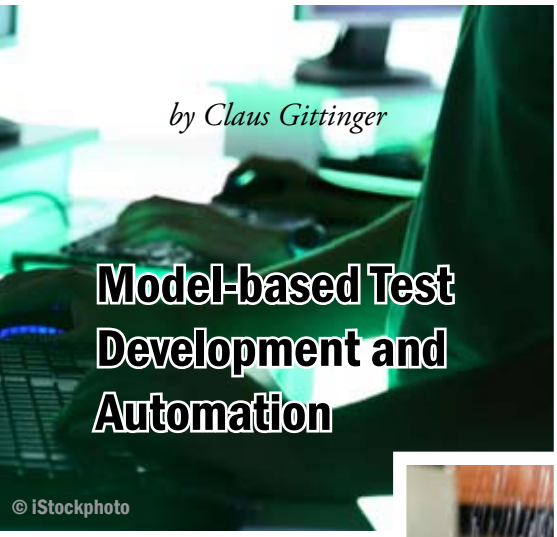
José Manuel Díaz Delgado

Contents

Boon and Bane of GUI Test Automation

by Mark Michaelis





Years of experience in providing professional services and in building R&D centers and testing labs all over the world, helped us to solve the equation and to be able to offer you end to end solutions of high quality, still using a competitive pricing scheme.

Some key facts about SELA:

Microsoft Gold Certified Partner	Accredited Training Provider - ISTQB
Leading the deployment of the new Windows 7 around the world	800 man-years experience in consulting and R&D activities
18-year track record	Locations: Canada, USA, Israel, India and Singapore

Worldwide professionals in the following fields and cutting-edge technologies:

Software Testing	**.NET**	**J2EE**	**Embedded Real-Time**
Application Security	**VSTS**	**C++**	**Agile and Scrum**

SELA can provide you services around the globe. Contact us, to start solving your equations.

s o l u t i o n s @ s e l a . c o . i l
w w w . s e l a . c o . i l / e n

Microsoft

SELA TECHNOLOGY CENTER

ISTQB

News

Industry Leaders Announce New Software Quality Testing Certification

PLANO, Texas – EDS, an HP company; the International Software Quality Institute (iSQI); and Rex Black Consulting Services (RBCS) today announced a new worldwide quality testing certification that establishes a new standard for the education of software testers.

The Quality Assurance Management Professional (QAMP®) certification helps technology organizations increase the effectiveness of, and achieve better business outcomes with, applications that meet business needs with reliable performance.

Product development, whether it is defining requirements or implementation, is no longer a centralized project as most companies operate development teams that work concurrently from locations worldwide. As a result, it is increasingly important to ensure that project teams are on the same page throughout the entire process.

"Worldwide standardization is the key to the success of quality testing services," said Stephan Goericke, director, iSQI, and inventor of QAMP. "QAMP provides a modular certification that bridges the gap between business analysis at the front end and test execution at the back end of the process. Moreover, year-long project experience is also recognized and independently evaluated."

"Widespread adoption of QAMP will help certified professionals effectively and efficiently work together to build quality software," said Rex Black, president, RBCS. "This will result in lowering the costs of field failure, increasing confidence in their releases and reducing the risk of defects."

Organizations today rely on software applications as part of their day-to-day operations, making software quality assurance critical for business success. Existing quality assurance programs typically focus on specific areas of the process, which creates an insular test and development process. To be successful, qualified testers have to bridge the gap between business and technology.

QAMP professionals must be certified and demonstrate standardized knowledge in requirements gathering, software testing and test management. To ensure that testers are trained on each step of the development process, QAMP incorporates several industry-standard certifications, including International Software Testing Qualifications Board (ISTQB) Foundation Level, ISTQB Advanced Level – Test Manager and the International Requirements Engineering Board (IREB) Certified Professional for Requirements Engineering – Foundation Level. The QAMP certification also requires two years of experience in software quality assurance.

"The new QAMP designation will differentiate professionals who have achieved the most comprehensive quality testing certification in the world," said Andy Mattes, senior vice president, Application Services, EDS, an HP company. "We are committed to having all of our test professionals certified in accordance with this standard and will also provide training courses for our customers. As a leader in quality testing certifications and application quality management software and services, HP is working to develop more effective methodologies and products that deliver better business outcomes from inception to delivery."

More information about the QAMP certification and training is available at www.qamp.org.

Testing performance in complex environments

by Markus Kröger

The leading provider of e-government and business application services to the German Chambers of Industry and Commerce (IHK) is the "IHK Gesellschaft für Informationsverarbeitung" (IHK-GfI). IHK-GfI enables its regional chambers located throughout Germany to perform all necessary tasks by operating a modern software portfolio. This includes several web-based applications and the IHK's MPLS-based wide area network (WAN) with more than 200 access points. In addition to the services delivered by IHK-GfI, each regional chamber has its own IT infrastructure and IT strategy. The IT services IHK-GfI provides must cope with this complexity and also integrate with all of the regional office settings. In order to objectively measure service levels and to prove optimization, IHK-GfI decided to introduce strategically positioned performance analyses and optimization measurement tools.

As a part of its analysis initiative, IHK-GfI sought a solution to both optimize IT service use cases that permanently showed low performance and to accurately reproduce sporadic performance problems. Further, they needed to identify overload situations in the back-end systems and integrate performance data into IHK-GfI's rapid development release management process. While the analysis software was primarily intended to be a troubleshooting tool and gather detailed and valid data to optimize internal software development, IHK-GfI also kept in mind the long-term objective of designing performance-based service level agreements (SLA).

Because of the specific structure of IHK-GfI's clientele, there were several requirements the new performance measurement software had to meet apart from sophisticated analysis functionality. In addition to coping with the diverse IT infrastructures of the IHK organization and enabling remote measurement in an easy way, the new tool was expected to be able to measure transactions across all system tiers involved and to consolidate information from reading points in the front-end and back-end including testing as well as production systems. IHK-GfI required a sophisticated decoding function and also expected the vendor to have network know-how as well as expert knowledge in software development. A vendor with expertise in both of these areas was particularly important for IHK-GfI since it was shifting its infrastructure from mainframe operations to three-layer architectures and significantly expanding its product portfolio.

After evaluation of several tools, IHK-GfI chose ApplicationVantage from Compuware. In comparison to other tools, it best met the organization's criteria. Additionally, IHK-GfI had previously hired a Compuware consultant to resolve performance problems and had seen both the capabilities of the solution and how to knowledgeably use it.

From an application point of view, there were several applications to be monitored. With its EVA product suite, IHK-GfI provides an enterprise resource planning (ERP) system which was specifically tailored to meet the needs of the IHK organization. Using the vendor-specific and proprietary T3 protocol, the system supports 6,000 clients and has currently an average of 2,100 simultaneous users. In addition to ERP, IHK-GfI provides CRM ServicePoint which serves more than 1,000 clients and an accounting system from specialist software vendor Diamant with more than 700 clients. In the back-end, Diamant is hosted on .NET servers and the ERP and CRM systems run on a BEA application cluster.

In this complex environment there were a variety of tasks for a performance analysis solution. For the EVA product suite, IHK-GfI decided to optimize end user performance in four phases by analyzing a wide range of use cases and taking into account their dependence on WAN parameters. The goal for phase I was to reduce the number of threads the services created from a range of 20 to 30 per action to 3 to 4. With regular analysis data from the network management team, software designers were able to identify the source of unnecessary threads and can now continue to optimize the solutions. In phase II the focus was on monitoring and reducing the bandwidth demands of each system. The objective of phase III was to optimize the query methods between systems and phase IV's goal was to maintain perceived performance of front-end systems. During the analysis it became clear that ApplicationVantage was not able to decode the front end T3 protocol automatically at the application layer level. However, measurement of application turns and bandwidth delivered a valid data set on which the analysis was conducted. Information from additional sources like the vendor and the development team also contributed to the analyses.

Individual services were also analyzed. For the CRM service, the performance measurement concentrated on web-services communication via SOAP. IHK-GfI found high volumes of data transfer which was caused by the description overhead the SOAP protocol produced and thus proved that SOAP optimization was necessary. The solution also measured the system's IP telephony capabilities for CTI integration and validated performance metrics. Even though direct optimization of the third party accounting software was not possible, IHK-GfI monitored the corresponding data flows and was able to provide valuable feedback to Diamant. To identify the sources of de-

lays on the client side, use cases that required sophisticated financial mathematics and therefore complex communication processes between application servers and databases were broken down.

IHK-GfI's next initiative is to increasingly use their performance analysis software as a proactive measurement tool rather than a troubleshooter to report on existing systems. The solution will become a standard tool for software development and be used in the quality testing environment. Over time IHK-GfI will expand its performance measurements and move from use case analysis and troubleshooting toward designing performance-based SLAs and a more holistic view of the entire IT environment.

Biography

As a physicist with his roots in natural science, Markus Kröger has been using scientific methods in IT for more than ten years. After starting with local network administration, he switched to e-mail administration and participated in a project which migrated an 8,000-client MS Exchange System to Lotus Notes/ Domino in 2002. At this stage Markus Kröger also founded a Notes User Group for the Lotus Domino Administrators of the IHK eMail network. Thereafter he became configuration manager (administrating SCM and build-systems) and concentrated on wide-area network and performance analysis/ optimization. In 2006, Markus Kröger led the project to switch the IHK's network technology from ATM to modern MPLS. His current work includes client and back-end performance analysis including monitoring and stress tests.

A brief introduction to testing technical embedded, software-specific systems

by Thomas Thurner

Software in technical systems typically processes sensor and switch signals, controls displays or actuators (motors, actuating elements) and thereby controls or regulates power and moments or triggers a chain of consecutive physical actions. These kinds of mechanical electronic systems are found either in production facilities or in the end product itself, e.g. in a car, airplane, train, or a medical device.

Essential requirements

Technically embedded systems are essentially characterized by the following properties:

- Real time control – not only the value area but the time area has to be controllable simultaneously as well. A correct value at the wrong time is just as faulty as an incorrect value at the right time.
- Robustness – here, for example, the compensation of tolerances in the controllable mechanical instrument or the compensation of temperature influences have a crucial significance. The recognition of age appearances in the hardware or surrounding mechanical devices are included here, as well as the handling of fuzzy decision thresholds in the event of noisy sensor levels.
- Security - under no circumstances may a danger be posed to humans or the environment by a malfunction of the mechatronic systems. It must always be possible to achieve a secure state under acceptable conditions. This condition is usually part of the system specification and guarantees an incremental shutoff or transition into emergency operation (a so-called "graceful degradation") in various error situations and while maintaining the essential partial functions for as long as possible.
- Hardware-specific software – the source code is dependent on the basic conditions of the hardware. Special microcontrollers are employed on control devices that have integrated a multitude of additional functions on the chip in addition the CPU. Examples of this include bus system protocols, special interrupt registers with automatic time recording, and the activation or input of analog signals with modulated pulse widths. This means that every microcontroller has hardware-specific properties and therefore a special compiler that includes extensive libraries for the implementation of these special functions. Therefore the source code can only be ported to other hardware in a limited fashion – it is not directly compatible with other microcontrollers under any circumstances.

Devices in the area of (mobile) entertainment, navigation or telephony/data transfer also have to be classified among the imbedded systems. Although they don't result in any mechanical activations, they are confronted with very similar challenges in terms of requirements for mass production, robustness and real time.

Prerequisites for the test of software for embedded systems

The functionality of an embedded system is realized by the software in an associated control device. The software for the control devices loads input values through sensors and activates so-called actuators (control elements, lamps, speakers, spark plugs, etc.) according to specific algorithms. However, the test of software for embedded systems does not just include the test in respect to the actual functional requirements but also the test of a high number of non-functional requirements. This means knowledge and verification on a system level – a verification on just a software level is definitely insufficient. Therefore the behavior of the control device always has to be tested in its overall system context.

But the system level adds another dimension of complexity – among each other, systems are often coupled through the physical environment and therefore have to harmonize with each other, even under contrary environmental influences. In a car, for example, an ABS therefore has to harmonize with the ESP, the brake assistant and the control of the drive train. An individual test of one of these functions is therefore insufficient - even if it proceeds positively all around. This means that in reality, the elementary brake functions, for example, are usually developed by one manufacturer due to their close interlocking and then integrated into one single component. But there are overriding functions in other components that build on each other. For example: a radar-based distance-regulating cruise control that has to harmonize with the ESP and engine transmission and is therefore networked with these components by means of data buses.

How is software for embedded systems tested?

For the test, the control device that has to be tested has to be activated in a simulation environment that reflects "its view of the world" to the control device. For this purpose, real data bus traffic is simulated for the control device and all electrical pins of the control device are stimulated in a digital or analog manner with correspondingly activated signal generators just as would happen during its real employment. In such a case this is referred to a hardware-in-the-loop – or HiL – test condition.

This "HiL" provides the tested control device (or a multitude) in real time with all signals of its real environment and simultaneously makes all corresponding electrical loads available so that real streams can flow as well. At the same time it measures all incoming and outgoing signal values of the control device.

The challenge for the "HiL" is that all signals for and from the tested control device also have to be provided, measured and recorded in so-called real time – otherwise the control device does not recognize itself in a real environment, may not find its starting conditions and thus does not even get activated, or immediately goes into a failure mode. This means that the semantics of all signals in the form of data values and their temporal phase condition towards each other have to match exactly from the perspective of the tested components or control device. For many applications this is a matter of precision in the microsecond range.

This environmental situation is usually carried out on a model base with the help of automatic code generation. Ideally this concerns already existing models that were employed in the development of regular or control algorithms of the control device. in the "HiL", these individual models then have to be integrated and synchronized into an overall environmental model.

If a user interface is part of the system, e.g. the displays or switches, robots can be employed for the operation of the switches and camera systems to check the displays. This makes it possible to have a high degree of automation during the test for the human/machine interface as well.

Before the HiL test station is employed, it has to be proven that it does provide a real representation of the environment - in other words, it has to be verified and validated on a system level. This alone is a task in itself that can not be addressed in more detail here.

Functional and non-functional test execution

Now the "HiL" can be used for functional tests. Test series can be executed around the clock and conspicuities can be recorded through a high degree of automation. The conspicuity symptoms can lead to causal areas that can then be examined in more detail.

Through the targeted manipulation of the input signals in the time or value area, the robustness of the tested control device can also be determined and documented. It is possible to simulate e.g. a noise and strongly fluctuating power supply, illogical signals with falsified contents and incorrect times or "slurred" signal edges and test which threshold values, error sizes or combinations thereof will lead to conspicuities and whether e.g. the system always achieves a predefined and secure state here.

In order to execute temperature influences or agitation tests, the tested components have to be positioned in climate-controlled closets or on so-called shakers, which of course additionally increases the complexity of the test. Even for EMC tests (EMC = electromagnetic compatibility), an HiL test station can be applied – but there has to be an assessment whether a static operation or the singular activation in just a few operating points is sufficient, so that EMC tests do not require the entire dynamics of an HiL functionality.

For larger systems, e.g. the integration of all control devices in one vehicle, an "HiL" has to simulate all relevant vehicle functions and their signals in a vehicle in real time – which also includes real electrical loads. This leads to large and complex testing stations that require entire lab rooms and generate an association to control stations in power plants, rail signal boxes or production facilities.
At the same time it should be remembered that this also requires driver and traffic or road models, which take the typical human operating factors from the layman to the expert into consideration and also cover the typical driving and traffic situations from the inner city to the highways.

Summary

This article intends to provide an insight into the world of tests for embedded systems and therefore does not address technical details. It shows that the testing of embedded systems can not be carried out on an exclusive software level, but that the tested control device always has to be observed at its system level, with all the real time requirements for a physical, analogous environment with special consideration of the achievement of a safe condition in case of an error.

Biography

Thomas Thurner finished his Diploma in Electronics Engineering end of 1988 and started his career 1989 at Mercedes-Benz AG as development engineer for networked electronic control systems. From 1991 to 2003 he had different project and department leading positions in the research division of Daimler AG concerning "Software architecture for vehicle electronic control units" and "fault tolerant electronics for x-by-wire systems". In 2004 he took responsibility as Director of the automotive software division of the Mercedes-Benz Technology GmbH. Since 2007 he is with SQS Software Quality Systems AG as Head of Business Unit "Industry and Engineering" and responsible for all services in the fields of technical embedded systems.

Honoury positions:

- President of the ASQF e.V.
- Member of the Advisory Board of the VDE Baden Württemberg
- Member of different conference program committees

testing experience

Díaz Hilterscheid

Testing Experience presents

Belgium Testing Day

February 23rd, 2009 in Brussels, Belgium

Schedule

	Track 1	Track 2
9:00 - 9:30	Check in	
9:30 - 9:35	**Welcome Note**	
9:35 - 10:15	**Key Note - Mieke Gevers**	
10:15 - 11:15	**Yaron Tsubery** ATP – The Factors behind Success	**Steven Mertens** Usability Testing
11:15 - 11:45	Coffee Break	
11:45 - 12:45	**Erik van Veenendaal** Risk Based Testing In Practice	**Geert Vanhove** When testing stops and politics takes over
12:45 - 14:15	Lunch	
14:15 - 15:15	**Alon Linetzki** Root Cause Analysis: Dealing with Problems, Not Symptoms	**Claus Gittinger** Model-Based Test Development
15:15 - 15:45	Coffee break	
15:45 - 16:45	**Geoff Thompson** Software failures – Can we learn from them	**Geert Lemmens** Agile: Testing at the speed of light
16:45 - 17:45	**Manu Cohen-Yashar** Security testing using free tools	**Stephan Goericke** Assure Quality – Keep Training Testers
17:45 - 20:00	Chill out / Networking	

Exhibitors

iSQI International Software Quality Institute

SOGETI

TOBIUS where professionals meet professionals

ps_testware Your devil's advocate

onetowin excellence in people & projects

exept SOFTWARE AG

Please register
by E-mail
register@belgium-testing-day.com
or by fax at
+49 (0)30 74 76 28 99.

600,- € 550,- €[1] 500,- €[2]
(plus VAT)

HOTEL SILKEN BERLAYMONT
Boulevard Charlemagne, 11 -19
1000 BRUSSELS
www.hoteles-silken.com

[1] for Testing Experience subscribers and ISTQB certified testers
[2] Early Bird (registration till 2009-01-15)

Interview Tauseef Khan

Tauseef Khan
Senior Director, Quality Assurance, Borland Software Corporation

"A big consequence of automation is that it doesn't find you any new bugs", Do you agree?

Tauseef: I would have to disagree with this statement. One of the goals of automation is to find regression defects sooner than later. Regression defects either are the byproduct of bug fixes or may be associated with any new feature implementation.

A well coordinated automation strategy can help find critical defects faster especially as part of regression test run. Our automated regression tests are kicked off automatically as soon as nightly builds are completed. This allows the development teams to respond quickly to the issues uncovered by the automated regression run. If the automation is not being updated to keep in sync with the new features added or new stories, you run the risk of making your automation obsolete.

Can you explain the positives and negatives of Automation?

Tauseef: The Positives of Automation are: Reduces test cycle and the defects can be discovered earlier with a fail fast strategy. Increases the overall productivity. Reduces overall cost of application development. Quicker turnaround of automated results reports with a more accurate interpretation of test execution. Encourages teams to test more often and more completely. Relieves constraints on resources - particularly cost and time. Improves the reliability and quality of the Product / Application. Better Test path coverage in terms of length and breadth. Aids in testing a large test matrix (different languages on different OS platforms). Automated tests can be run at the same time on different machines, whereas the manual tests would have to be run sequentially unless additional resources are deployed. With end to end automation, most regression tests can be run on a nightly basis unattended allowing teams to focus on fixing the defects in the upstream of application development life cycle.

The Negatives of Automation are: Test automation are susceptible to failures more often due to lack of proper understanding and thus can discourage the teams. Highly repeatable automated test cases can minimize the chances the chance of discovering all the important problems. It is important that the goals of automation are clearly understood by the team. It might end up costing more than the manual testing if poorly planned.Can't automate visual references, for example, if you can't tell the font color via code or the automation tool, it is a manual test. This is more of a constraint than a negative of automation.

What should be considered while calculating the ROI value for automation?

Tauseef: When calculating ROI for automation you need to look at the real value automation is giving you. The ROI value is not the value of automation vs. the cost of executing these tests manually, but is instead the benefit of this type of testing, plus the benefit of whatever the manual tester is doing while these simple tests are executing.

It is important to consider the following factors when evaluating the ROI of automation.

1. The cost of executing the automated tests. Consider the infrastructure required.
2. The cost of maintaining the automation as the product evolves release by release.
3. The cost of any additional tasks necessitated by the automation
4. ROI will vary depending on your business environment and the application under test.

Let me explain in detail – The real return in investment for automation is based on each type of automation you choose and what value it adds to the overall testing effort. Think of performance testing. The value of performance testing is readily realized and is not based on the cost of automation. Moreover, automation is the only practical way to perform these types of tests.

Similarly, each type of automation will have its own unique return and it may not relate to manual testing activity. It will be made up of both tangible, where automate testing saves 50 man hrs and intangible ben-

efits, where automation is done as a parallel task to development and this in turn helps developers with their unit testing, and can catch bugs sooner in the SDLC.

One of the problems with the classic example of ROI for automation is that it only looks at one type of automation, regression scripts. There are however many types of test automation. Example:

1. Load and Performance testing
2. Installation and configuration testing
3. Boundary testing
4. Code Coverage and runtime analysis
5. Compliance testing
6. Smoke and unit testing
7. Localization testing
8. Regression testing
9. Etc….

Do you feel automation can be achieved and maintained by manual test engineers?
Tauseef: This is an interesting question – It all depends on the skill set level of the test engineers. If a manual test engineer lacks programming skill, it will be very difficult to achieve test automation. Most test automation tools incorporate the use of a programming language and while most tools have been designed to make writing functions simpler by incorporating easier-to-use interfaces, the end result still ends up looking like what it really is: program code. Basic skills set requirement for an automation test engineer is to have the skill and experience in creating and maintaining a large number of reusable modules and test scripts. He/she will need to have a good understanding of basic programming language techniques as well as structured programming concepts.

Having said that, automation over the number of years has been evolving and has gone through from early days of record and play to approaches like data driven and key-word driven methodologies. One of the new approaches that is creating a buzz in the industry is the Business Rules-Based Test Automation Approach. If implemented correctly, this approach has the potential of removing the dependency from the skilled automation expert.

To summarize, it is important to have a clear understanding of the organizational goals and align your testing strategy to your business goals and plan accordingly.

Important: Don't think of test automation as a "project" that has an end. Instead, compare it to a "product" that will be built and maintained. It should live as long as the application under test needs testing.

How automation can be practiced in Agile?

Tauseef: Software Product development methodology trends today see us leaning more and more towards Agile practices. Software Test teams therefore, need to re-align to this new approach.

As you know, the biggest difference of agile methods from traditional waterfall is the short feedback loop. The whole concept of agility in essence is no more, than "build the most important piece, evaluate, adjust, and repeat". Automated tests in the agile methods serve as a very important tool for shortening the feedback loop. This new shared responsibility methodology requires both process changes and new tools to support that process. The process change is QA's involvement in development from the beginning, working along-side development. As we know automation requires proper development methodology, and therefore requires planning and time to develop. Moreover automation requires the product to develop into a state of stabilization or maturity.

When Agile teams are developing the object model allegory, automated testers should be using the same time to develop an automated testing allegory which will be used to create a design approach to test scripts so they can later be put together for regression. Automation engineers should reuse the unit tests created by developers, and make sure they are executed and maintained and are regularly added to the regression suites.

As sprints/iterations are planned, time for development of underlying infrastructure for automated tests should be included in the stories.

Automation in Agile requires a quick turnaround so the tasks should include: data creation scripts, scripts which expand on developers' test driven development examples and expand the testing through data driven techniques. Automation testers should look for ways to abstract code into functions and external data sources to allow for quick changes to the execution steps and drive expected results as well as input and output data.

What would you recommend to organizations that are faced with the challenges of improving quality? Are there any best practices that can help achieve maturity in quality?

Tauseef: This is a great question. Undoubtedly, most organizations are facing key challenges of improving quality and reducing time to market with limited resources.

Like you and so many of our customers, my team is under constant pressure to test more with less. And with the current economy, I don't see anything changing for the better any time soon.

However, as a commercial software vendor with 25 years focused entirely on helping customers build better software, I have the benefit of some great experience and expertise from which to draw. Borland's experience culminates in not just a great set of software solutions, but in a deep understanding of what it takes to consistently improve quality and reduce costs in a demanding, dynamic and competitive market. We have found—from our own experience as well as from that of our customers and partners—that improving quality is a journey requiring a good plan, a firm understanding of where you are and distinct goals for where you'd like to be.

That's why; Borland has developed a whole new practical methodology named appropriately as "The Quality Maturity Curve (QMC)". This unique, new approach provided us and our global teams with the big picture of how we can mature our testing practices incrementally, based on our unique objectives in a tailored, unique way.

Unfortunately, due to limited time in this interview, I wouldn't be able to get into all the details of this compelling approach. It probably requires a follow up dedicated interview just on this subject, however, let me share some critical phases of the quality maturity journey that we went through.

The Quality Maturity Curve consists of several dimensions:

First, and most foremost, it identifies 5 stages of quality maturity.

1. Testing is the first step any organization takes to assure projects are not catastrophic failures. Most projects get some level of testing, so this is the first stage - unstructured and very often poorly managed.
2. Test Automation is the stage where organizations begin to introduce functional and performance test automation. It represents acknowledgement and desire to better utilize time and human resources to provide more test coverage and to introduce some structured management to the process.
3. Quality Management considers the needs of test teams to really employ structured management processes to improve how testing gets done and how quality is presented to management.
4. Operational LQM is the next step beyond just providing great testing practices and visibility. It reaches further "upstream" into the developer testing realm and spans the IT organization with integration to requirements and software configuration management areas. This is where quality across the software lifecycle really begins paying dividends.
5. Enterprise LQM, for the largest or most sophisticated, quality-driven organizations is the pinnacle of the quality maturity curve. At this point, an organization's Application Lifecycle Management (ALM) practices are well-defined and working. Economies of scale, centers of excellence and standardized tools and skills are delivering measurable and fully repeatable quality for every critical project.

The important point here to note is that maturity is relative to your organizations' goals and often, size. With the possible exception of Traditional Testing, all other stages represent a set of best practices which deliver increasing benefits and efficiency based on how many teams and organizations or departments are interacting. For some organizations, typically small ones, Test Automation may be the ultimate goal, where for others, setting goals of Enterprise LQM make the most sense, but achieving that stage requires making incremental progress to assure the right objectives are set along the path. So, while organizations may stop at any stage along the way, it is not really possible to bypass any stage without losing the value and benefits that each stage offers – with the possible exception of Test Automation.

By the way, Borland is now offering a free QMC profile for organizations that are interested in tackling the challenges of improving quality. You will be surprised that in less than 8 hours of interactive sessions, you can have a complete picture of where your organization is at in regards to the quality maturity curve and where it needs to focus to go to the next level of maturity.

Profitable, beneficial deployment of test automation in the area of SAP Core-Banking

by Alberto Vivenzio

Introduction:

The topic of test automation can no longer be ignored, particularly when it comes to the introduction of company-wide software solutions. In addition to cost and efficiency improvements, the main emphasis is here on the aspect of a measurable quality improvement.
In particular for the introduction of standard software solutions to cover the core business or the core processes of the enterprise, test automation will play a strategic role.

What is generally meant by automation?

In general, automation means the conclusion of recurrent functional processes. In a software environment corresponding tools are deployed for this purpose, so that automated test processes can be executed. The results, irrespective of whether the execution was successful or not, are then documented in an execution protocol. In general, the aim is to execute the software without human interaction.

What advantages does this bring?

The automation of processes brings the project a certain staff independence. This is useful during the test phases of a software project. After the software has been implemented, it must be tested with respect to its suitability and correctness. It must be proven that the business processes will continue to work also after the deployment of the new software.
In software projects, three parameters are always critical: time, cost and resources. In addition, the quality assurance is immensely important in banking projects. Nothing is more fatal than situations where incorrect or erroneous data is made available to a customer. Based upon experience, one has to invest the same effort for quality assurance as has been invested for the implementation.This is where test automation can play a role.

There are various testing tools available, each of which has its advantages and disadvantages. In the end, the selection of a tool for a specific project is partially a subjective decision. In some projects this decision does not need to be made, since the customer already provides appropriate tools. In SAP projects, it is also very simple. You simply use the tools eCATT and the Test Workbench from the Solution Manager provided by SAP.

What is eCATT?

With eCATT (Extended Computer-Aided Test Tool) SAP offers a tool, which is an integrated component of the SAP base components, and which is therefore free of charge. In addition to recording and replaying of transactions, this tool contains various additional features for the testing of applications and program sections:

- Direct read and write operations to the database
- Calls and execution of functional components and BAPI's
- Execution of ABAP within the scripts
- Test of transactions and reports
- Test of applications, which are executed outside of the SAP GUI for Windows and SAP GUI for Java (by means of an interface for third-party providers)

Besides this, eCATT is embedded within the „Test Workbench". The „Test Workbench" is a complete program suite for temporal and hierarchical management of test plans, test case catalogs and test cases. It supports the test manager as well as the testers in managing the entire test scenario, at one central „point of truth".
Via the information system of the „Test Workbench", ad hoc status regarding the test activities and test results can be obtained at any time. Results and status must no longer be manually transferred to other documents.

What does this mean in practice for a SAP banking project?

Let us view this in the light of the „isadem bank ", an internal project at isacon group for the implementation of a demo bank, i.e. a bank, which can be shown to interested decision-makers from financial services organisations. It is based on the SOA- compliant platform SAP Banking Services 6.0 and in its first release offers the complete functionality needed for deposits!
If one implements a core banking system such as the „isadem bank" as a green field project, it means that one does not replace a legacy system and therefore does not have to migrate its data into the new SAP system, Decisions are therefore only required in two areas. On one side one has to answer the question „Which processes would make sense to automate?". On the other hand one has to clarify how to fill the „isadem bank" with test data, so that a meaningful demonstration will be possible.

Which tests will make sense to automate?

At the implementation of an SAP solution, less is programmed compared to a conventional software development process. The various modules already exist and are adapted to the customer's needs through the customizing process. The largest programming effort is needed at the creation of the interfaces and the implementation of customer-specific requirements which have not been implemented in the SAP package itself.
In general, there is a high degree of correspondence between the customer requirements and the SAP Deposits Management system. The customer selects a core banking system and - as the name already suggests - the „core functions" of a bank are the main requirements. These main requirements will be implemented

through the so-called standard test cases. Apart from smaller adjustments, these test cases can be reused from project to project.
The reusability of the standard test cases is an aspect for possible automation. In general one can say, that the more frequently a test case is executed, the more beneficial its automation.
The automation must first be developed. In most cases, this is done with the help of software, in our case eCATT. The process is recorded and displayed as a test script within a source code editor. Subsequently, the input values are entered, which shall be tested, parameterized, and then replaced by placeholders. Thereafter, various input combinations can be applied by means of these placeholders via a TXT file. In the case of eCATT, the creation of test scripts and the pertinent test data is separate. The link of these two is created in the test configuration. Furthermore, various SAP systems can be connected. The systems are set up in the module System Data, and are brought in contact with the test data and test scripts in the module Test Configuration.
eCATT is a user-friendly tool which offers the option of inserting instructions via list functions. Nevertheless, programming knowledge is also helpful.
It is also possible to record whole sequences or business processes and then replay them at the push of a button. This, however, makes the maintenance of the scripts more difficult. It is instead recommended to proceed in a modularized way where each component is captured in its own script. Subsequently, sequences can be created and via a masterscript the particular modules can be called.
In order to answer the question, whether tests should be automated, data was gathered during the isadem project. Based on this, a model was developed which compares the effort for manual and automated test cases.
In this context, the existence of the standard test cases is an important basic condition. Besides this, the average values for the execution of a manual test case are set to 20 minutes. For logging and processing of one defect found, 15 minutes are estimated. Compared to this, 180 minutes are estimated for the automation (i.e. development) of test cases, for their execution 5 minutes and for the handling of errors also 5 minutes. In addition, 10 minutes are used as a buffer for the adaptation of the automated test cases to other projects.

Repetitions / Defect	3	4	5	6	7
0	4,9	3,4	2,6	2,1	1,8
1	3,8	2,8	2,3	1,9	1,6
2	3,1	2,4	2,0	1,7	1,5
3		2,1	1,8	1,5	1,4
4			1,6	1,4	1,3

Or:

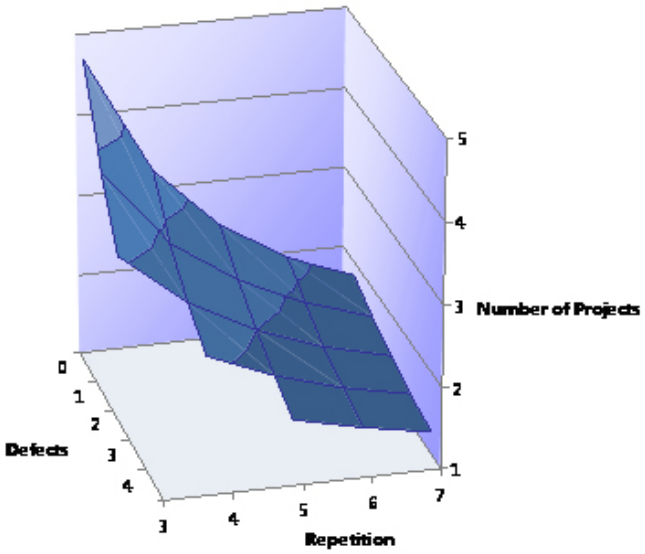


Table 1: Number of projects[1]

The result area of the table corresponds to the number of projects within which the automated test case shall be used, in order to match the effort needed in the manual case. This table is based on average values and assumptions which are derived from experience gained in earlier projects. Furthermore, only one test case was used, which is a stark simplification. In case of a sixfold use of each test case per project and in case of one defect arising per test case, two projects would be enough to gain an advantage in terms of effort spent.
At first glance, one might assume that this table does not support the argument for automation. However, for the execution of a manual test case human effort is needed. The real question is: How long can a tester execute one test case after the other without losing quality? In case of automation, however, the developed scripts can be run after the working day and the results reviewed at the beginning of the next working day. These considerations are difficult to express in numbers, and were therefore also not included in the calculation. This does, however, illustrate how quickly the time lead of the manual test cases is disappearing in the case of multiple execution.
Automation costs. On one hand time for the development and on the other hand for the resources required for the acquisition and introduction of the software. These negative aspects are to be weighed against the above-mentioned advantages, before one decides to automate the test execution or not.

How do I obtain test data?

The test of SAP Deposits Management is based on the business partners. In the case of the isadem bank a business partner can be a person, group or organization, whereby a business partner is distinguished - amongst other things - by the following parameters: Name, first name, street, postal zip code, place of residence, country, federal state. Each business partner can again assume different roles:

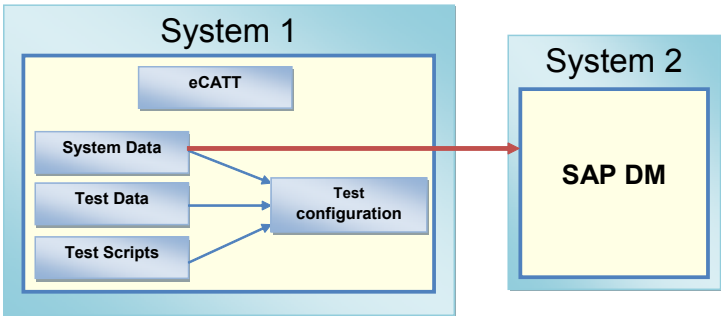
Roles:

- Business partner general
- Account holder
- Card holder
- Correspondence recipient
-

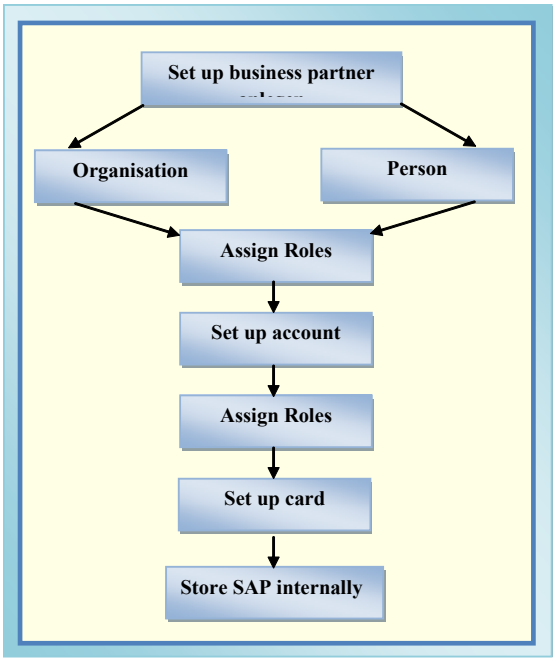
Depending upon the role assigned, actions can then be performed for each business partner: Set up account, set up card etc.
Meaningful business partner data is created using a VBA/Microsoft Excel-based tool based on various combinations, and is entered using a generated eCATT test script. If the test cases are executed manually, then this database can be the starting point. In the case of automated tests, the developed scripts are executed in accordance with this process.

The simulation of the banking business

For the isadem project, the simulation of the banking business is a core task. It shall simulate business processes under approximately realistic conditions. This serves as an illustration of the processes within SAP Deposits Management. The objective is to simulate funds transfers, deposits, disbursements over a period of time. The customer is then given the option to withdraw money using a graphically represented ATM or to print account statements. The simulation is also implemented using eCATT. This is based on the following process:



[1] Patrick Haibach, Diplomarbeit: „Testautomation for SAP Deposits Management“, 12/2008



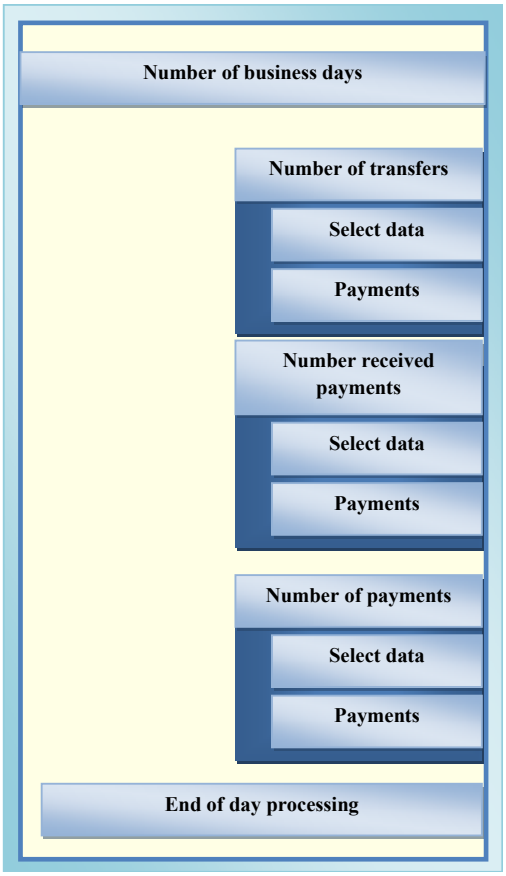
1. Master Script

The functional module „Store SAP internally“ stores the business partner, account and card data in an SAP-internal data base table.

2. Master Script

Before the script is executed, the number of business days to be simulated must be specified, and in addition to this also the number of transfers, deposits and disbursements per day. The functional module „Select data“ selects random data out of the internal data base table (1st Master Script) and enters them into each input screen. With the start of the end of day processing, the business day ends and a new day begins. Depending on the number of business days specified, the execution may run over a long period.

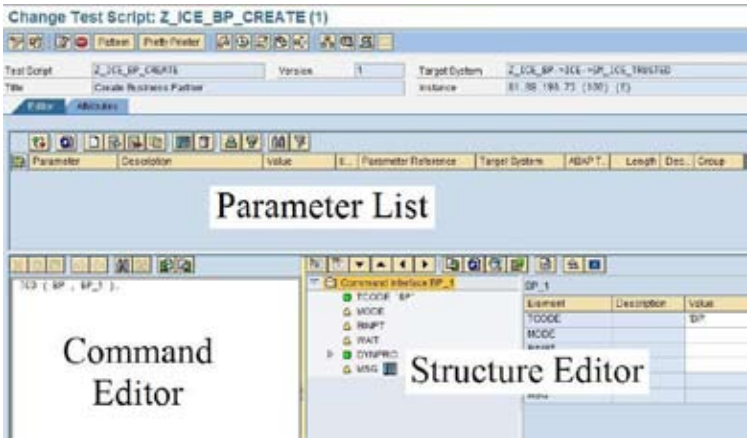
The purpose of the acceptance test is that the customer tests the requested product himself. This shall of course not be done in the production environment. The simulation described above can therefore serve to represent a production system.



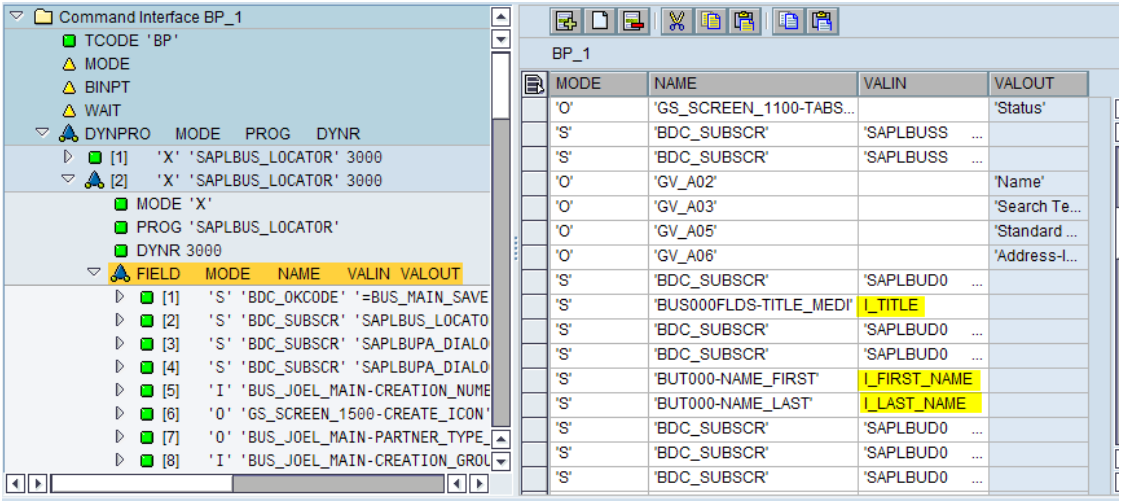
Automation with eCATT (Example)

The automation process starts with the recording of the transaction to be automated. At the start of the recording, a new SAP window is opened. This will show the recording. Subsequently, the user returns to the eCATT window and terminates the recording.

Depending on the set-up of the recording and the granularity, only one line may be shown in the code editor (Command Editor). The individual steps are then shown in a tree in the structure editor (Structure Editor). Within this tree, the parameterization will also be performed.



For the parameterization, the applicable input screen is selected. The entries made during the recording are then converted into placeholders (Parameters). In this example first name, surname and title are parameterized. The parameter names are I_TITLE, I_FIRST_NAME and I_LAST_NAME. The “I” stands for “import parameter”. When the recorded script is executed, these parameters are used to enter arbitrary values.



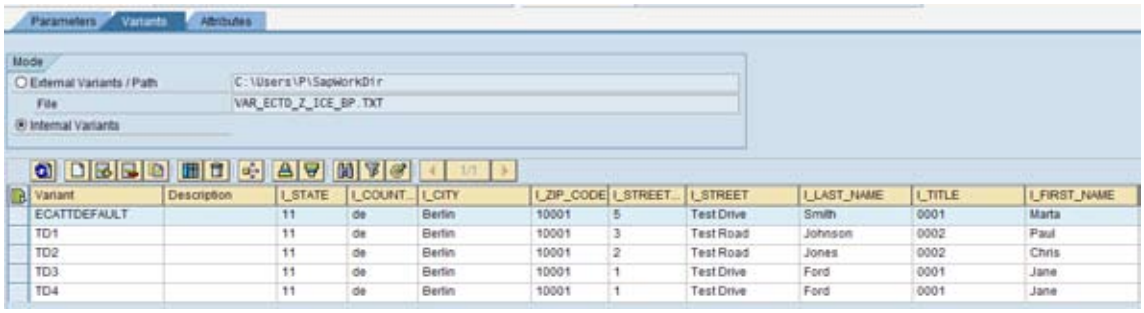
The parameter list contains all parameters which were set up. The column „Value“ shows the input value which was used during the recording. This value can be changed.

The parameter names make up the interface to the test data. The test data can only be assigned to the parameters if the names themselves are the same.

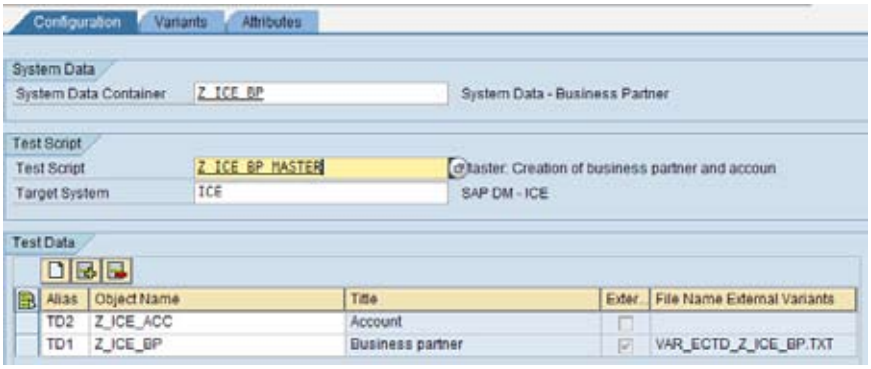
Parameter	Description	Value	I/...	Parameter Reference	Target System	ABAP T...	Length	Dec...	Group
E_BP_NUMBER	BDC field value		E	BDC_FVAL		C	132		
I_STATE	State	11	I	REGIO		C	3		
I_COUNTRY	Country	de	I	LAND1		C	3		
I_CITY	City	Berlin	I	AD_CITY1		C	40		
I_ZIP_CODE	ZIP Code	10001	I	AD_PSTCD1		C	10		
I_STREET_NR	Street number	5	I	AD_HSNM1		C	10		
I_STREET	Sreet	Test Drive	I	AD_STREET		C	60		
I_LAST_NAME	Last name	Smith	I	BU_NAMEP_L		C	40		
I_TITLE	Title	0001	I	AD_TITLETX		C	30		
I_FIRST_NAME	First name	Marta	I	BU_NAMEP_F		C	40		

The test data can be entered manually into a table or added by means of an external TXT file. However, this TXT file has to follow certain conventions so that it can be processed by eCATT.

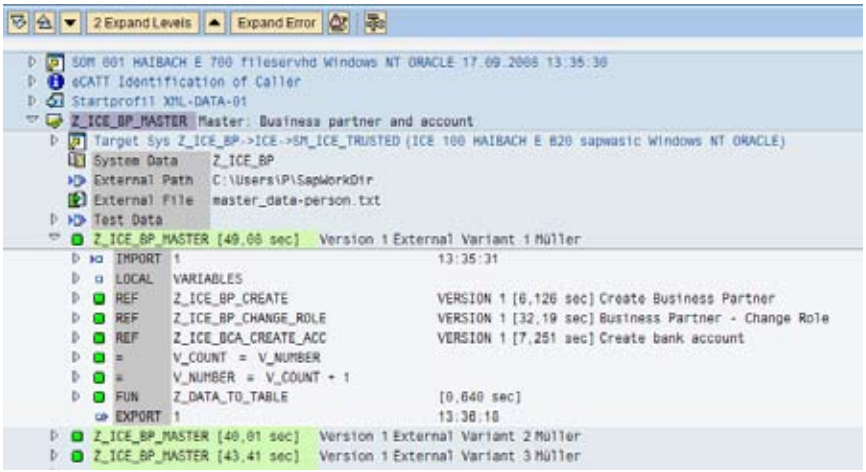
Test data and test scripts are combined by means of the test configuration.



At the execution, a certain system is used and within this the test script is processed. Depending on the settings and the test data entered, the script may use different test data at each iteration.



Each run will be shown in the log as an extra node. If a step terminates with an error, it is marked red.



Summary

As one can see, test automation is complex. It needs time and resources. In the case of the isadem project, we have focussed on test automation, which is executed via the SAP GUI. This reduces the cost for the test tools by 100%, since the SAP tools used are included free of charge. If there are also other Front Ends (e.g. a teller cash or teller application), via which tests are automated, then one could consider the deployment of further tools. There is a sufficiently large choice of good and user-friendly packages available. However, these do not come free, and the license costs must also be considered in the cost calculation.

With our many years of SAP banking experience, we have created a package of standard test cases, which with little effort can be used in every SAP banking project, and we have automated these test cases.

We will continue to develop isadem in several releases. Release 1 deals with the deposits business. For Release 2 (for example), the complete lending business (Loans Management) is added. For each new release, we can again rely on our automated test cases and in doing so clearly reduce the test effort for each release, whilst at the same time attaining high application stability and quality, since the automated test cases represent our internal quality gate for all projects.

Biography

Alberto Vivenzio is at present Managing Director for the Competence Center – Quality Assurance and Authorization at isacon group.

In his team he has a large number of experienced, ISTQB-certified consultants, who possess excellent expertise, in particular as test managers and test professionals.

Alberto himself has for more than 15 years been successful in the area of software test for core-banking applications. During this time, he was responsible for the creation and implementation of test strategies at numerous banks world-wide, in particular in the area of SAP Core Banking.

Lately he could successfully demonstrate his expertise in the area of Next Generation Banking at a newly incepted bank in the Middle East.

Exploratory Testing Tutorial

by James Lyndsay

March 2-3, 2009 in Amsterdam

Limited places

Getting a Grip on Exploratory Testing

Exploratory Testing is a discipline. Focussed on the just-delivered system, it exposes real-world problems more rapidly and more reliably than approaches which concentrate on prior expectations. It forms a powerful complement to scripted testing, and is especially powerful when matched with the large-scale confirmatory testing found on agile projects. Good exploratory testers find better bugs, and give fast, clear feedback to their teams.

Many testers are able to work without scripts and explore a system. However, most take a single exploratory approach, and so become less effective when they have exhausted that approach. Many teams use exploratory testing, but do not know how to manage it effectively.

This course allows participants to experience a range of exploratory techniques in a disciplined framework that will allow exploratory testing to be managed and integrated with existing test activities. The course is built around hands-on, brain-engaged exercises designed to enhance and reinforce the learning experience.

Participants will discover:

- The test design skills to probe a system and trigger a bug
- The analysis skills to model the system and understand a bug
- The discipline to manage their exploration and sustain their bug rate

Course Description

This two-day course is a comprehensive guide to exploratory testing, with exercises deigned to engage testers of all abilities, and structured workshops to help participants use and share the lessons learned. The course will introduce participants to a wide variety of approaches to exploratory testing – parsing the system, systematic exploration, modelling for success and failure, questioning, attacks and exploitations. Starting with three basic exploratory frameworks, we will construct models of the systems under test, and use those models to enhance our testing and to judge problems found. We will develop attacks and exploitations to reveal deep risks, and look at the potential risks in target technologies.

Throughout the course, we will use session-based testing to support the personal and team discipline necessary to support effective exploration, and will examine ways that ET can be managed within existing processes.

This tutorial will be of greatest use to test analysts, senior testers and test managers, but will also be immediately relevant to designers and coders. Direct experience in exploratory techniques is not necessary, but delegates with one or more year's experience of hands-on testing will get most from this course.

1250,- EUR
(plus VAT)

testing experience
Knowledge Transfer

Please register by
email kt@testingexperience.com
or fax +49 30 74 76 28 99

The Record & Playback Fairy Tale

by Koen Wellens

"Once upon a time, there was this nice little tool that could automate test cases with a simple record and playback action." It could be the beginning of a wonderful fairy tale, but not a realistic one. Test automation is more than just record & playback. It is an entire process with its own challenges: aligning people with different points of view, avoiding common pitfalls and creating a solid structured approach. Let us take a closer look at the path of successful test automation.

Record & Playback

What does "Record & Playback" mean in the world of test automation? A test tool captures user actions in record mode, such as a mouse click or a keyboard stroke. When the recording is stopped, the resulting script contains several steps to simulate the captured user actions. If you click on a "Play" or "Run" button, the tool executes the script and all your recorded user actions will replay automatically. This will probably work fine during the first execution, but what if the script is played against a new release or an error occurs? Moreover, you can certainly not call this a test, as you did not include any validation at all!

Let us take a closer look by using an example. The application under test is a flight reservation system. The following tests need to be automated:

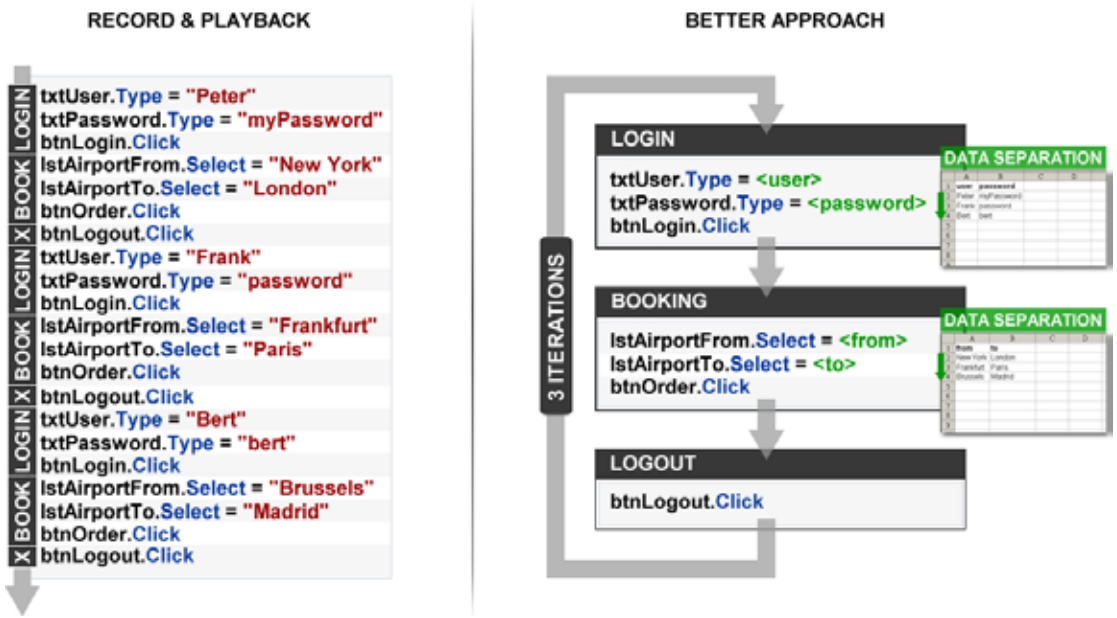
- Log on using 3 different accounts;
- Book 2 flights with different airports for each login.

Record & Playback approach

Press the record button, login manually, book 2 flights, logout and authenticate again using the second login. Book another 2 flights. Perform this action once more and stop recording. Now you have automated the given test cases.

A better approach

Press the record button, login manually, book 2 flights and logout. Stop the record mode. Now you will iterate the recorded steps programmatically by using a loop. Next, replace the static data in the steps (user name, password and selected airports) by a dynamic value. A dynamic value is a value retrieved from e.g. an external data pool (spreadsheet, database, flat file) at run-time. Finally add steps manually to verify that you have logged in successfully and that the correct ticket price is displayed.
Now your script is automated to run the same tests, but with 1/3rd of the code compared to the Record & Playback approach. Additionally, you have added validation and separated data from the script (called "data-driven test-



ing"). It is obvious that the second approach is easier on maintenance, more robust and last but not least: it has finally become a real test by adding validation.

Different people, different perspectives

The term "record & playback tool" is misleading as you can do a lot more with these tools. You can add validation, separate data, connect with a database, etc. The term leads to a troubled view on test automation. A tool vendor tends to stress the ease of use of the record & playback capabilities, a manager looks at test automation as a simple record & playback task, lowering resources and time allocation to set up automated tests. A test automation engineer will soon learn that record & playback is not enough, requesting more time and resources for the automation process.

Estimations of workload and time allocation will change during the process. In practice, there are two common mistakes:

1. **A test automation tool is an extra tester**

A common mistake is that test automation replaces the manual tester by an automated version, the virtual tester. This is not the goal of test automation. The real ROI is the quality improvement of your application through the increase of test coverage, avoiding human errors and speeding up test execution. Manual testers can focus more on the validation of new functionalities instead of spending time on regression tests.

A test automation project requires permanent resources. The projects need to be set up and maintained. A normal process is that the automated regression test set keeps on growing throughout the project life cycle. After all, tests for new functionality become regression tests after the release. These tests can be automated for the next release.

2. **Automate everything**

Another mistake made using test automation tools is that people will try to automate as much as possible. However, not every test is suited for test automation. Regression and other iterative tests are suited for automation, whereas non-iterative tests are not.

Automating everything is counter-productive. After a while, the test automation engineer will have so much maintenance on the existing scripts that the return on investment (ROI) vanishes. You do not benefit from automating non-iterative tests. The code cannot be reused, but still needs to be maintained at each release. If you have a complex test that needs to be executed once for each release, it is not worth automating. The time to automate and maintain this test will always be longer than the manual execution of it. As it is a complex test, you need to create exception-handling code to uncover possible errors.

After a few releases, the scripts become non-maintainable and the automation project is stopped as it has resulted in a bad ROI. If the project gets a second chance, the automation engineer will start all over again, defining the correct scope, thoroughly analyzing the automation project, setting up a structure (framework) and using the experience gained.

Five steps for successful test automation

Many test automation projects do not have a high success rate after one year. Why is that? There are many potential pitfalls:

- The ROI is not tangible. How do you measure the increase in quality?
- Miscalculation in work effort: record & playback appears to be harder than expected
- Non-structured approach resulting in a chaotic and non-maintainable automation project
- No scope is defined: do not automate everything!
- Lack of tool training and experience.

Test automation should be done systematically in a structured approach. There are five steps for successful test automation:

- Planning
- Preparation
- Proof of concept
- Implementation
- Maintenance

Planning

Each phase in the project must be planned together with the project manager, test manager, tester, automation engineers and possibly a steering committee.

You should have a clear view on the current testing methods and information of the infrastructure. Gather information by interviewing key people who have experience in the field of test automation.

Preparation

During the preparation phase, you should define a pilot project and select the test cases that need to be automated. Define the roles and responsibilities and prepare the input data (data management).

Proof of Concept

The test automation tool must be configured in order to be compatible with the application under test. The tool must be able to capture user actions. Since more and more application types are being used, the configuration part is not as easy as it seems. Application types can be Web, ActiveX, .Net, Java, SAP, Siebel, …

It is possible that the test tool will not recognize the application "out-of-the-box", which means you need to configure the tool, server or even the application before user actions are recorded.

Next, you need to automate a limited amount of test cases to prove that the application can be automated. Select an easy, a normal and a complex test case. Demonstrate the execution and reporting of the automated test cases to the decision makers.

Implementation

If the proof of concept is successful, the automation of the selected test cases (scope defined at preparation step) can start. Analyze the selected tests, think about data separation, functional decomposition (separate your code into reusable functions), reusability of certain business components.

Modularize your script into clear-cut building blocks. Finally, you can press the Record button and put method into practice.

During this phase, you can set up a test automation framework. In brief, frameworks can vary from documentation on tool usage (naming conventions, standard approach) to a full-scaled framework based on a spreadsheet or database layer. These complex frameworks are often project-dependent and require a lot of effort at the start of the automation project. Documentation on standardization is an absolute necessity. A layer-framework can be useful, depending on the project. In general, tests are stored in the layer part (e.g. spreadsheet) and can be maintained by manual testers. These tests are written in keywords, making a clear overview of what needs to be tested. The manual testers can even prepare the automated test before a new application release, based on the release notes. The keywords of the layer part are interpreted in the test automation tool by a script engine. The downside is that this engine is very complex. If structural changes are required, the complexity of the script can increase maintenance time.

Maintenance

A test automation project expands together with the application under test. A new release may offer new functionality that needs to be automated. The existing automated scripts need to be maintained, new automated tests need to be added and other persons will take over the automation project in the future.

You have to prepare for these events by providing documentation for each script, by constantly allocating the proper resources for the project and by training newcomers in the tool, the script and the execution.

Happily ever after

Most fairy tales end with the phrase 'they lived happily ever after'. Whether your test automation story has a good ending or not, depends on certain criteria.

Create awareness amongst all players.

Misleading terms like "Record & Playback" soon lead to underestimation of workload and preparation time.

Start with a structured and phased approach.

When there is a solid basis, maintenance becomes manageable and you will benefit in the long term. After all, a test automation process has the same life cycle as the application under test. So get it right from the beginning!

Start by planning the project.

Gather information, manage your data and, last but not least, make the right selection of test cases. A proof of concept can evaluate if test automation is possible on the project. After a positive evaluation, the implementation phase can begin.

Structure your scripts

Modularize and separate data, decompose code into functions and document everything.

The maintenance phase determines the good ending of the automation story. Will the scripts prove to be robust against new releases and will maintenance be manageable?

I hope that your automation story ends happily ever after!

Biography

Koen Wellens is a senior test consultant at ps_testware nv.
Koen holds a bachelor degree in Applied IT and has worked as a qualified technical tester with 5 years of experience in the field of test automation, performance testing and tool support & coaching. He is a tutor in test management and automation tools on a European level. Koen gathered experience in various sectors, such as retail, finance and government. Today Koen is responsible for both Quality Center and QuickTest Professional Life-cycle Management, including support, training, customization and coaching.
Koen has an excellent knowledge of HP/Mercury and IBM/Rational testing tools and is both ISEB/ISTQB-certified and HP-certified Product Consultant.

Boon and Bane of GUI Test Automation

How to get the most from GUI Test Automation

by Mark Michaelis

Synopsis

In recent years the importance of well-designed User Interfaces has increased a lot. If nothing else, it is the rise of Web 2.0 applications, the applications for everyone. Nowadays User Interfaces have to deal much more than before with untrained people sitting in front of their computers.

So it is no wonder that not only the "automation behind the scenes" (Unit Testing for example) gained in importance, but also the automation of User Interface Tests with all its boon and bane.

This article describes the advantages and disadvantages of automatic GUI testing and tells you some lessons I have learned during my career as GUI tester. You will also get to know some selected tools I used for automated GUI Tests. A very personal insight into my work rather than a good comparison of all available GUI testing applications.

Bane

Let me start with the Bane of GUI Test Automation. This was the first contact I ever had with automatic GUI tests. The developers of the company I worked for before were well aware of the importance of GUI Tests, but they also knew of their problems. They used the Netbeans Jemmy Module.

Jemmy

Jemmy is no capture and replay tool, a quite common approach for GUI Testers nowadays. You develop your tests just as JUnit Tests in Java Source Code. This means you are actually blind when developing your tests. This has some disadvantages:

- There is no (implicit) additional testing by the author of the test.
- The author of the test has to know Java (in this case), although he does not need to have this knowledge from the perspective of his domain.
- Maintainability very much depends on the capabilities of the author. This is true as well for the identification of GUI components and for the actual error messages.

The result was obvious: Once developed, the tests were hard to maintain and it was rather hard to find anyone willing to dig into the code.

Nevertheless, experiences at this company paved my way into Quality Assurance. I changed to CoreMedia AG, Hamburg where I had the opportunity, no, the challenge to maintain GUI Tests for another Java Application, created with HP WinRunner, formerly known to be developed by Mercury Interactive.

WinRunner

Unlike Jemmy, WinRunner is a capture and replay tool. This means that while recording you actually see your application and you have an additional test through this for free. WinRunner does an important step: It separates the Map of the User Interface from the actual tests. This results in a database describing your GUI elements like this:

```
FileOpenWindow {
       class: window,
       label: Open
}

FileLoadButton {
       class: button,
       label: Load
}
```

Now the tests will refer to the GUI elements via this map. This has two advantages:

1. Test execution script and GUI Maps can be maintained separately. As a consequence: If only the layout changes, you only have to adapt the GUI Map at one specific location.
2. Depending on the keys (here: FileOpenWindow, FileLoadButton) you choose, your tests might be easier to read.

I will return to this example later. Because it is actually a boon, not a bane. The banes about the tests I adopted were:

- The GUI Maps were actually too rich. Think of a button which should be at coordinate 100,345 and has a label "Load". If one attribute does not match anymore, the test will fail. It would have been enough to just say that the button has the label "Load". That was enough to make the button unique.
- The test execution is described in a proprietary language called TSL (=Test Scripting Language), which in itself has some disadvantages:
  - No developer of the application (implicitly) knows about this special language.
  - GUI test developers again had to dive into a different non-GUI medium.
  - There is only little information about this language you can find e. g. in forums.
- The test scripts were not written to fail. In the case of an error, they did neither give clear statements of what failed or why, nor were they able to leave the application in a stable state so that other tests could continue.
- Tests depended on the results of other tests. This is actually no WinRunner problem, but I got aware of this while debugging the scripts. With one failed test at the root, all successors also failed of course.
- Tests were not rerunnable. Just think of

a read-only file to be created at a specific location. If you rerun the test, it will fail because the file already exists. While this is no problem during automatic test execution, it is a problem when debugging: Testers always have to reset their environment before a test can be rerun.
- Common actions (like "create document") occurred, step-by-step, at multiple locations in the scripts. As the creation steps in the application slightly changed, all of them had to be adapted
- WinRunner is for Microsoft Windows platforms. And most Microsoft Windows applications are not Java applications. This bears (at least) two other disadvantages:
  1. WinRunner cannot run on multiple platforms like MacOS, Linux, etc.
  2. WinRunner has severe problems, especially when it comes to supporting newer releases of Java.

Boon

Discussing the boon of any automatic test first starts at the ROI, the return-on-investment. Shura from Sun has written an interesting article about this topic titled "Automation Effectiveness Formula". Shura sets the investment to be equal with development time, which of course is not true in every way, but it is a good start. Shura points out that there are many variables to take into account when talking about effectiveness:

- The only alternative to automatic testing (if you want to test at all) is manual testing.
- Tests have to be run at least on every release.
- Tests have to run on all supported platforms.

Manual Testing

Shura already stated that the only alternative to automatic testing is manual testing. The scenario I know is that you have a couple of test plans which describe step by step what to do on the user interface. Testers replay these test plans and mark the different test steps as failed or success.

This should be done on all platforms the application supports, and it must be done on every release. This already rings a bell of warning, but let us first face the advantages of manual testing:

- It is obvious that the tester and the customer have the same view on the application. So it is a perfect acceptance test.
- Testers have no problems dealing with changing GUI layouts between versions. And if they have problems, their customers might stumble across them, too.
- Testers can do more explorative testing besides the actual test plan.
- GUI testing can be easily outsourced, as the testers do not need to know more than the customer.
- Manual GUI tests can be done by "Eating your own Dog Food": Use your product. This is the best and cheapest test you will ever get.

However, there are also disadvantages:

- Each release, each platform multiplies the costs.
- Test cases tend to explode easily. You have to manage carefully and use appropriate selection schemes to find the right tests.
- Manual testing between releases (nightly tests) is impractical. So you have no early reports on possible problems. Therefore, time has to be reserved for bug-fixing after the GUI tests are completed. In most cases, this will be short before the actual release date.
- Testers tend to become blind. If one and the same tester executes a test case multiple times, he will very likely start to skip test steps or at least not check the surrounding application not directly involved in the test path. This is especially true if he does the same just on different platforms.
- Manual tests are most likely to be skipped when the project runs out of time.
- The tests are not reproducible, as testers tend to forget the actual steps they have taken until an error occurred. Only recording the manual steps might help here.

Automation

My experiences with Automatic GUI Testing are more like a boon than a bane. Still, there are some WinRunner tests which need to be maintained (and they really break with each new release). They are the bane I am trying to get rid of. The boon started with a reboot from scratch.

This reboot was initiated by a new GUI testing application which was recommended to me. It is QF-Test developed by Quality First Software. This tool finally formed my expectations of a good GUI testing tool.

Expectations of the perfect GUI testing tool

1. Capture and Replay
   This means that you can press the record button, do some clicks and then replay these clicks by the tool. This is perfect to use in short-time projects with a short maintenance phase. It is inappropriate for product development, as your application will change over time and maintenance of such captured scripts is time-consuming. However, you can still use Capture and Replay to get to know the testing tool's language better.
2. Test Scripts saved as text
   This might also only be important for product development, and if a version control system is used. For binaries (such as commonly used Excel tables for data-driven testing) you will have no support to see differences.
3. Testing Language offers Try/Catch/Finally statements
   This is important so that you are able to react to exceptions which might break your tests. If you don't know such a statement:
   - "Try" will contain the code to be executed.
   - "Catch" will contain the code to handle exceptions, such as clicking away error dialogs.
   - "Finall" will contain the code which will always be executed irrespective of whether the try-code was successful or failed with an exception. Typically, some cleanup is achieved. For the CMS Editor tests I wrote, it closes for example documents I opened during the test.
4. GUI elements are recorded as mappings
   This is what I mentioned before: In the testing code, only references to the keys exist. Keys which point to the identifying descriptions of GUI elements.
5. Screenshots can be made
   Ideally they are automatically made in case of failed tests. However, it would be sufficient if you have the chance e. g. to make a screenshot in the catch statement. This really makes the first error analysis much easier.
6. Test scripts can be modelled in a GUI
   This is important because the GUI Testers can use their domain knowledge (of GUIs) to write the tests. And if you can choose your statements at the GUI, they are just easy to learn. All this is possible, because actually most steps for GUI Testing (about 90%) are quite simple and do not require a rich language. Another possibility is to use FitNesse, which allows a high abstraction level from the actual test execution steps. The tests themselves are modelled in a Wiki and are nearly as easy to read and understand as the customer-readable test cases.
7. There is a non-proprietary fallback language
   If there really should be a case where the language elements of the GUI testing tool are not sufficient. you need a fallback language. In case of QF-Test, it's Jython for example (a mix of Java and Python). These scripts should be easy to hide in the test scripts modelled in the GUI, so that only a small selected group of testers need to know about these scripts. Non-proprietary, because it is always good to be able to get help from communities on the web.

Of course, there are many more expectations, such as the quality of the generated reports, availability of a debugger or that tests can be started from the shell (for automatic nightly tests). But these seven checks I would take with me the next time I had to evaluate a GUI-Testing Tool. Needless to say that QF-Test passes all these checks with ease.

Restart from Scratch

The restart from scratch was important and easy. We had existing WinRunner tests which already covered some aspects of the application under test. This gave us time to concentrate on a new concept and on the learnings from previous GUI test banes.

- **GUI Maps:** All *needed* GUI elements got collected into extra files which could

Are you doing *automation* the smart way?

Companies look at test automation as the cure-all for their problems in improving testing cost and effort. By itself, automation cannot solve your problems.

Automation involves a significant up-front cost for building the automation test script suite. For this effort to generate sizeable returns, the scripts created need to be reusable and amenable for any user to execute them. This can only be ensured if the automation objectives are defined in advance and the entire automation exercise is well planned.

The right approach to test automation

Obtaining a clear understanding of the automation landscape before commencing automation is crucial to delivering high returns on your automation.

An Automation Assessment will establish consistency around your automation approach across different teams; thereby reducing the development and maintenance cost by enhancing reusability. The assessment will help you standardize your automation in terms of process, framework, tools, metrics, operating model, environment and test data management. In addition to an evaluation of your current automation practices, incorporating mechanisms to analyze existing manual processes for automation readiness will maximize your automation benefits at an enterprise level.

Defining your automation scripting standards and guidelines and designing a reusable user-friendly automation framework will help you build high-quality test scripts with consistency

Cognizant makes it simple

Cognizant's expert group on Test Automation has developed a wide range of frameworks and tools that will help you get more returns for your automation.

Automation Assessment Cognizant's automation experts have developed a comprehensive framework to assess your test automation across a wide range of parameters and sub-focus areas.

ROI Calculator This proprietary tool provides an estimate of the ROI before the automation project commences. This will help you take an informed decision on whether, and when, to automate your test scripts.

CRAFT Cognizant's Reusable Automation Framework for Testing is designed to simplify the process of script development and execu

C2AUTO Cognizant's C2Auto helps to identify the ideal test cases for automation.

To know more about Cognizant - the world's largest testing service provider, please log on to www.cognizant.com/testing

Cognizant | Testing Services
Passion for building stronger businesses

World Headquarters
Cognizant Technology Solutions
500 Frank W. Burr Boulevard
Teaneck, NJ 07666
Phone: +1 201 801 0233, Fax: +1 201 801 0243
Toll Free: +1 888 937 3277
Email: inquiry@cognizant.com

be easily shared between all tests. Only the ones needed, this is important: Unused mapped GUI elements just blow up your files and once you need them it is most likely that they won't match any existing component anymore. And instead of relying on coordinates, component hierarchies etc. I gave every tested GUI element a name via setName(). A nice anecdote is that even a partner of CoreMedia got aware of this when he had to test extensions to the GUI. He was very pleased by this fact, told it to one of our developers who then told it to me. As you can see QA processes sometimes have surprising results.

- **Framework:** To get around duplicated code, I decided to build up a framework. The goal was to create a framework to get as fast as possible to the test case you want to execute. If you need some kind of document to write to, no matter what it is: the framework will offer a function for this. The result is quite obvious: Nowadays I can create many new test cases within minutes. Not much more time than a manual tester would need – for executing it **once** on **one** platform.
- **Facade:** For some checks the old and the new tests had to dig deeper beneath the GUI and access the API of the application directly. In the old tests this had not been made obvious to the application developers. And when developers changed the API, the GUI tests had to be adapted too. For the new tests I bundled this GUI Test API into a Facade. On refactorings with some IDE support, they will be automatically adapted to internal API changes. On the QF-Test side, I created a stub for this facade which made using the facade inside QF-Test rather transparent.
- **Fail-safe:** Especially for newly created regression tests, it was ensured that they failed with unfixed versions and that they left the application in a stable state.

Web-Test-Automation

I have to add this section, as I mentioned web testing as an important requirement for the future where web applications are increasing and even Office applications can suddenly be run within your browser.

At CoreMedia we are currently using Sahi which allows to write the tests in JavaScript, which of course has the benefit that you have full access to your web application. The problem of Sahi is that it is easy to write unmaintainable code, as there is no concept of GUI-Maps. But with some coding guidelines this can be enhanced. They include such things as to bundle access to GUI elements and keep them separate from the test scripts, and to write the top-level-code in a way which makes it easy for a human reader to see the relation to the user test cases.

So a test like this:

```
enterSearchTerm("Sahi");
submitSearch();
checkResultsContain("Sahi");
```

is easier to read than a test like this:

```
_setValue(_textbox("q"), "Sahi");
_click(_submit("Google Search"));
_assertNotNull(_link("Sahi"));
```

(taken from an example provided by Stringy-Low in the Sahi Forum).

A good alternative seems to be Selenium, which uses FitNesse at highest level to describe the tests. It was recommended to me quite often. And I am currently keenly following the further development of QF-Test, as they will also support web testing with the new 3.0 release.

Conclusion

Automation of GUI tests is a bane if you do not respect some rules, particularly with regard to maintainability. However, it is always a boon compared to manual GUI testing which simply costs too much money if the tests are executed regularly and perhaps even on multiple platforms.

My recommendation: Evaluate the application you want to use to write tests carefully, and carefully design your tests.

I hope I could give you some insight into the rules I found for myself, which you can find in the box "Mark's Rules for GUI Testing Automation".

Bibliography

- JavaWorld: Automate GUI tests for Swing applications by Ichiro Suzuki <http://www.javaworld.com/javaworld/jw-11-2004/jw-1115-swing.html>, 2004-11-15
- NetBeans: Jemmy Module <http://jemmy.netbeans.org/>; last release January 2006
- Automation Effectiveness Formula by Shura <http://jemmy.netbeans.org/AutomationEffectiveness.html>
- Fitness Acceptance Testing Framework <http://fitnesse.org/>
- QF-Test by Quality First Software <http://www.qfs.de/>
- Sahi Web Automation and Test Tool <http://sahi.co.in/>
- Selenium – Web application testing system <http://selenium.openqa.org/>

Mark's Rules for GUI Testing Automation

These are the rules derived from the boons and banes of this article. These are my very own rules – but you are free to adopt them.

1. Write your tests so that they are readable from top-level like a use case. This makes it easier for others to understand your tests, to get an overview of the use cases covered, and of course it eases the debugging in case your tests signal a failure.
2. GUI-Maps: Never address GUI components directly. Always use some kind of mapping.
3. Do not be too specific in describing your GUI components: Less is (often) more.
4. GUI-Element-IDs: If your application's development language supports setting aids for components, use them and tell your developers to set them. For Java you should use setName() for example, for Web applications the ID attribute.
5. Write "Tests to Fail". Be aware that your tests will fail sometimes. And then they have to deal with it: Report the failure, end the test in the best possible way and pave the way for the other tests to follow. In short: Cleanup!
6. (Of course) do not make tests dependent on the results of other tests.
7. If possible, create tests which can be run multiple times without the need to reset your application.
8. Use Capture & Replay only in short-time projects or for getting to know the testing application.
9. Use Data Driven Testing whenever possible.
10. Don't store any automation control statements in binary objects such as Excel tables.
11. If you ever need to access the API of the application from the tests: Use a facade!
12. Build up a framework to cover common actions and to hide them from the actual testing code.

Biography

Mark Michaelis is ISTQB Certified Tester and Software Engineer Quality Assurance at CoreMedia AG, Hamburg since 2005. His main focus is on automation of GUI Tests and automatic setup of test environments.
After studying computer science at the University of Ulm and writing his diploma thesis on the automatic navigation of robots through reinforcement learning, he started as Java Developer at H.U.T GmbH in Hildesheim in 2000.
In 2003, he changed to Gentleware AG, Hamburg developing on Poseidon for UML (a UML modelling tool) and got deeper into testing automation through JUnit and Jemmy.

Practical approaches to improving your testing by maximising code coverage in complex database and SOA environments

by Huuw Price

Introduction

All testers and some developers grapple with the problems of testing as much of an application as possible with the minimum amount of effort. This article discusses some practical approaches to stressing as much code as possible while keeping the effort level to a manageable level. I would like to thank Richard Bender for his input which inspired me to try and tie together disparate methodologies into a practical end-to-end approach to increasing code coverage.

Most testers have experience of test automation tools such as QTP, Facilita etc. and to a greater or lesser degree managing the data inputs and outputs from complex tests. Some testers have exposure to code coverage toolsets, while all are familiar with the concepts. Many academics, (indeed it seems to be the main research area of a large part of academia), work on code coverage tools and analytic techniques to improve software design, see http://crest.dcs.kcl.ac.uk/ run by Dr Mark Harman of KCL. However, there are few companies that have an end-to-end integrated approach to maximizing their code coverage during testing.

Before continuing, it is worth noting a few practical issues:

- Most applications are built using disparate modern and legacy technologies, for which code coverage tools do not always exist.
- Setting up code coverage can be time-consuming and may not be worth the effort.
- Specifications and documentation are not always of a high standard.
- Testing is usually done at the end of a cycle and is not embedded into the development lifecycle.

This paper will cover some of the techniques and tools to build the most accurate input to tests and also the practical methods and tools of creating the data to run these test. Simply building an optimized set of inputs is just the start of a solution; testers need to be able to expand these tests into real data in complex databases and SOA environments.

White Or Black Box Testing?

Code coverage assumes you have access to the code and can see all the paths through the code. Black box testing reverses this in that you only have control over the input and the ability to monitor any output. Code coverage tends to be used in component testing, which is where you have discrete sections of code with defined input and output. In a more typical test scenario, larger groups of components need to be tested as one unit, and with the advent of complex SOA environments complete end-to-end tests need to be designed and - a harder task - implemented. In addition, testers need to balance the need for complete coverage with only limited time to test.

From a practical point of view, having access to complete code gives you very useful information that can be used in designing tests. However, a balanced approach needs to be used for both black and white box testing. From a tester's point of view, it's all just testing with more or less information.

Trillions Of Combinations

Hardware engineering tends to have far greater success at eliminating failures than software: The reasons for this are not for this article. However, one of the techniques used very successfully is to optimize the test inputs. A simple group of on/off switches that can be set in particular orders quickly grows to trillions of combinations; hardware engineers have developed techniques to eliminate redundant tests, identify key relationships and make the very large numbers small. The application of some of these techniques to software means that the amount of code exercised can be significantly increased, whilst keeping the combinations of test inputs to a minimum.

Cause And Effect

One very useful methodology, as outlined by Richard Bender, is to identify cause-and-effect actions which show where combinations of input cause either explicit actions or intermediate "nodes" to be set. The identification of these cause-and-effect graphs are an effective way of optimizing tests inputs.

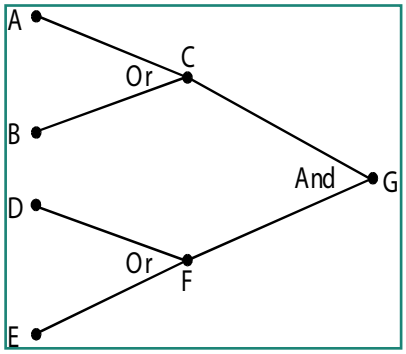


Figure 1- Cause- Effect Diagram

In this case, nodes F and C can be turned on by A or B, or D or E respectively, so varying combinations of D and E and A and B will have little effect on result G. These nodes can be identified within specific programs or at a higher level within the application. For example, a customer may have a low credit score; setting this as a discrete node as part of the test input rather than having to set up multiple customers (some of whom have a low credit score) will reduce the test inputs for upstream testing.
The identification of what these cause-and-effect relationships are is based on clear unambiguous specifications. A more subtle and difficult area for the tester is to identify which of these relationships are important and which ones to ignore. The tester will also have to decide which of the many data inputs need to be included as "core" in the testing. Some data elements may not be that important and can be left out.

Requirements Parsing

While access to the code is a useful way of identifying these "nodes", clear requirements are essential to building test inputs. Ensuring that specifications are clear and unambiguous is essential. A simple AND or OR out of place or not clearly defined can cause confusion all the way along the development lifecycle. In the code coverage example below, a comma out of place could create an ambiguous definition.

> If the customer is a business client or a preferred personal client and they have a checking account, $100,000 or more in deposits, no overdraft protection and fewer than 5 overdrafts in the last 12 months, set up free overdraft protection. Else, do not give overdraft protection
> *Code Coverage – Overdraft Protection Example*

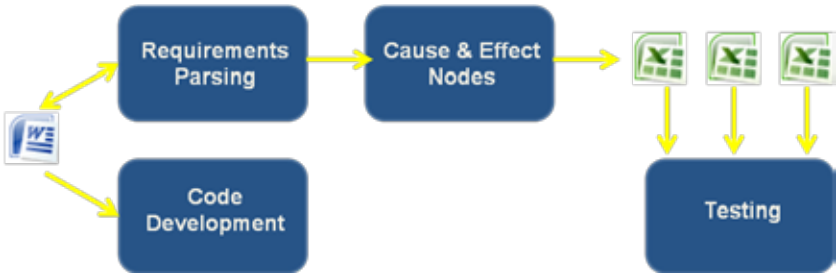
Coverage Worked Example:

If a specification is not clear and is not validated prior to development the following can happen:

Developer Guesses Correctly	Tester Guesses Correctly	Passes Testing	Correct Code in production
✔	✔	✔	✔
✔	✘	✘	✔
✘	✔	✘	✔
✘	✘	✔	✘

Figure 2

A very useful technique to validate specifications is to use test parsing and analysis tools that can quickly identify ambiguous definitions early on in the development lifecycle.



The input from the clear text parsers can be validated and is a valuable resource as input to test case generators such as Bender-RBT.

Building Test Input

Once the requirements have been clearly defined, the design of the test inputs can begin. There are numerous algorithms, tools and services that can help with this task. I will use our overdraft example and compare four different methods:

- All code combinations
- All Pairs
- Jenny
- Bender-RBT

A strong contender that I did not include is the web service http://www.testcover.com. I would recommend testers to take a look at this site and sign up for a trial, as they offer a service that is continually being updated with the latest academic research.

Tools such as Bender-RBT allow the synthesized output to be defined graphically; for example, here is the parsed text input to the tool:

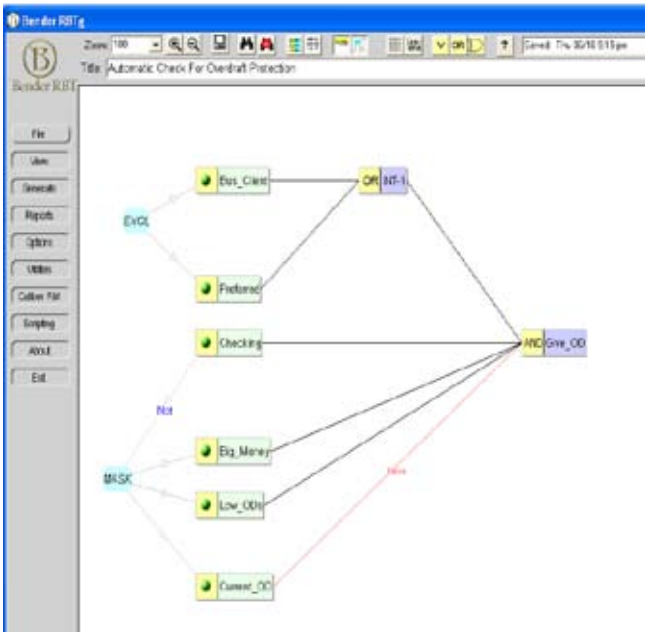


Figure 3 – Bender-RBT cause-effect builder

As you can see, the clear specification of AND and OR is easy to identify and verify. The identification of intermediate nodes is crucial to the creation of smaller sets of test input.

The output of the RBT tool will create a set of tests that can be used as input to test the process.

		TEST#1	TEST#2	TEST#3	TEST#4	TEST#5	TEST#6
Causes:							
Bus_Client		T	F	F	T	T	T
Preferred		F	T	F	F	F	F
Checking		T	F	T	T	T	T
Big_Money		T	M	T	F	T	T
Low_ODs		T	M	T	T	F	T
Current_OD		F	M	F	F	F	T
Effects:							
INT-1	<OBS>	T	T	F	T	T	T
Give_OD	{obs}	T	F	F	F	F	F

Figure4 – RBT definition and coverage matrix

VARIATION	TEST#1	TEST#2	TEST#3	TEST#4	TEST#5	TEST#6
1	X			X	X	X
2		#				
3			#			
4	#					
5			#			
6		#				
7				#		
8					#	
9						#
Unique vars	1	2	2	1	1	1
Total vars	2	2	2	2	2	2

As you can see, the tests take advantage of the INT-1 intermediate node to reduce the set of tests. In addition, the production of an expected result will reduce the time for checking any results.
Once the test inputs have been defined, they can be fed into tools such as Datamaker to generate the physical data either as input to capture replay tools, directly into databases or as flat files which are fed into applications.

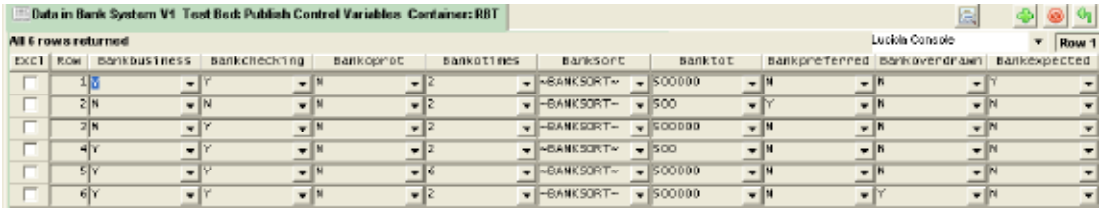


Figure 5 – RBT test cases captured in Datamaker ready to generate the physical data

Test data input was generated using three other methods:

All Pairs

All Pairs or pairwise testing is a combinatorial testing method that for each pair of input parameters tests all possible discrete combinations of those parameters. Tools such as Datamaker will automatically generate these combinations for you.

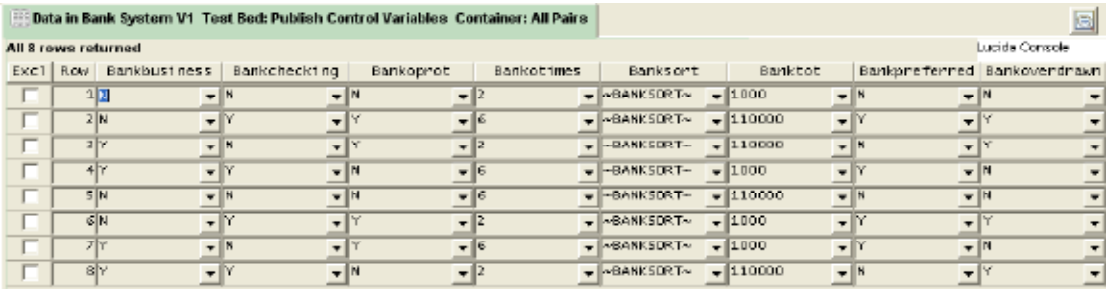


Figure 6 –All Pairs test cases captured in Datamaker ready to generate the physical data

Jenny

Jenny is a tool for generating regression tests. It will cover most of the interactions with far fewer test cases. It can guarantee pairwise testing of all features that can be used together, and it can avoid those feature combinations that cannot.

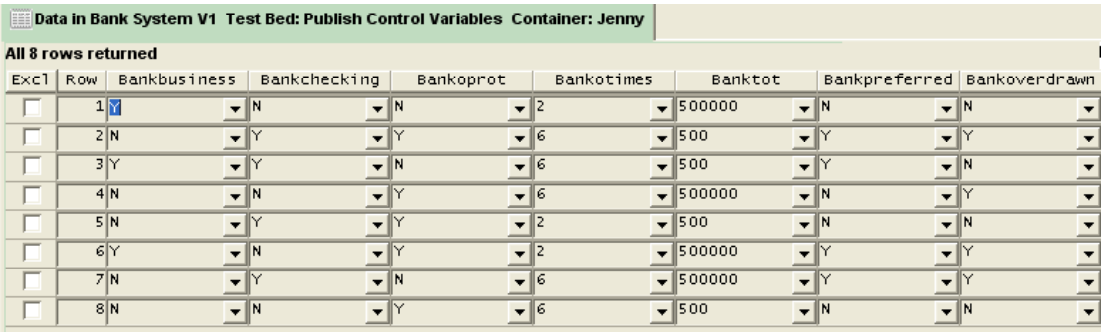


Figure 7 –Jenny test cases captured in Datamaker ready to generate the physical data

All Code Combinations

All combinations of variables are defined. In this case this is manageable. However, if you add a few more variable, the number of combinations will grow rapidly.

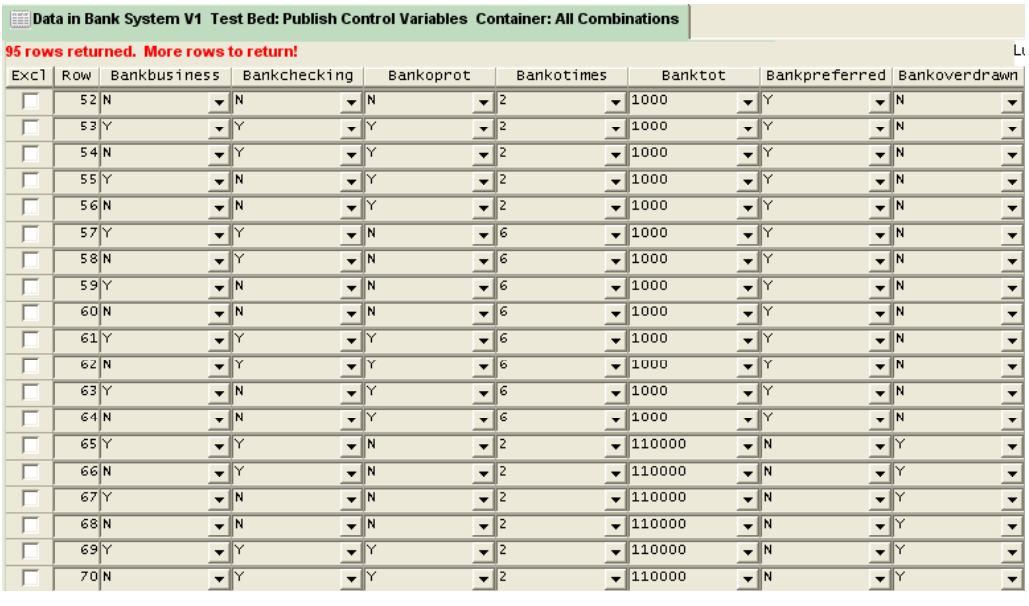


Figure 8 –All code combinations generated and captured in Datamaker ready to generate the physical data.

Building The Physical Data

Once you have decided on your test cases, you need to prepare the physical data to test your application. In our example, the data is held in an Oracle database and is spread across multiple tables. However, the factors for the production of data, whether by directly entering it into an application via tools such as QTP or creating flat file input to the SOA process, is the same.

As the program we are testing is a batch program, the tester needs to build the data to test the code. The data must contain the combinations of attributes we have identified from our test analysis. These vary from 6 test cases for RBT to 128 for all combinations. As a tester tasked with testing our code, you are immediately confronted with the problem of how to create these test cases.

Creating The Data Using The Application

You could enter the data by hand for each test case. The problem with this is time, creating an account and entering transactions such that it has been overdrawn over five times in the last year could be a very complex task. A similar level of complexity could exists for all the variables. In addition, the testers would then have to understand how to use the application in areas they are not familiar with. In our example, the Credit Control testing team would have to run the banking batch transaction system to simulate overdrawn transactions.

Searching For Data That Matches Your Criteria

Many testing facilities simply copy production data into development and testing environments. Nowadays this opens up all sorts of data protection issues. Nevertheless, it is still common practice.

Once you have existing data, you can construct queries to search for the combinations of data that match your test cases. If you are lucky, you will find them all very quickly. The problems with this technique are that you are in effect re-coding the code that the developer has created to track down the data. For complex queries this may be difficult, and the queries may be slow due to the size of the database. In addition, it is likely you will not find all of your test cases. By its very nature, increasing the code coverage takes you to areas in which existing data are very rare, as the majority of production data is very similar.

Hacking Existing Data

A very common technique is to find an account holder who is close to our first test case, then go in and edit the data to match the first criteria. This process can then be repeated by either running the batch program each time or checking the result, or by tracking down similar account holders and directly editing different account holders, one for each of our test cases.

In our example, the seven variables are spread out across six different tables. In the real world this is likely to be larger.

SQL #1 (16) | BANK_CUSTOMER (1) | BANK_ACCOUNT (2) | BANK_ACCOUNT_TYPE (1) | BANK_ACCOUNT_OVERVIEW (12)

SQL | Data in BANK_ACCOUNT_OVERVIEW | Status | Plan

All 12 rows returned

Bo Id	Bo Date	Bo Ba Id	Bo Max Bal	Bo Min Bal	Bo Transaction Activity
2	2008-08-19 00:00:00	1	6359.5	-2543.8	12719
3	2008-07-19 00:00:00	1	14950	5980	29900
5	2008-06-19 00:00:00	1	9930.5	3972.2	19861
8	2008-05-19 00:00:00	1	15066	6026.4	30132
10	2008-04-19 00:00:00	1	14007.5	5603	28015
12	2008-03-19 00:00:00	1	6028.5	2411.4	12057
13	2007-08-19 00:00:00	1	13689	5475.6	27378
14	2008-02-19 00:00:00	1	4530.5	1812.2	9061
16	2008-01-19 00:00:00	1	17208	6883.2	34416
18	2007-12-19 00:00:00	1	2264.5	905.8	4529
20	2007-11-19 00:00:00	1	13627.5	5451	27255
22	2007-10-19 00:00:00	1	4247.5	1699	8495

Figure 9 –Customers have multiple accounts of different types with a summary of monthly activity.

Generating Data

This is by far the most effective method of creating perfect test data. The advantages are that you guarantee that the data is as you need it, and the next time you test you already have test data that can be used for the next set of tests with slightly different combinations of data criteria.

For our case, we follow a few simple steps:

1. Copy a customer into the Datamaker repository. This will be used as the basis for the generation of all required test data cases.
2. Create variables for which you need to pivot the data, i.e. variables that can control what changes as the data is generated. In our case, we have created variables that match directly with our test case design.

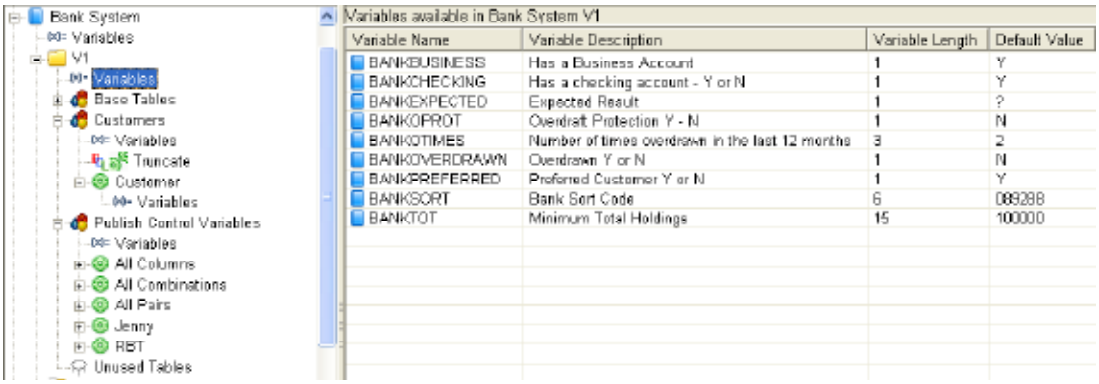


Figure 10 –Create substitution variables for each test case variable which can be changed when each test case is generated.

3. For each of the variables, change our data template (this is the original copied customer that has been turned into a generic data object) to

affect the columns in the data that control our desired data effects. For example, if the variable BANKOVERDRAWN is set to Y, place a '-' (minus) sign in front of the column BA_CURRENT_BALANCE.

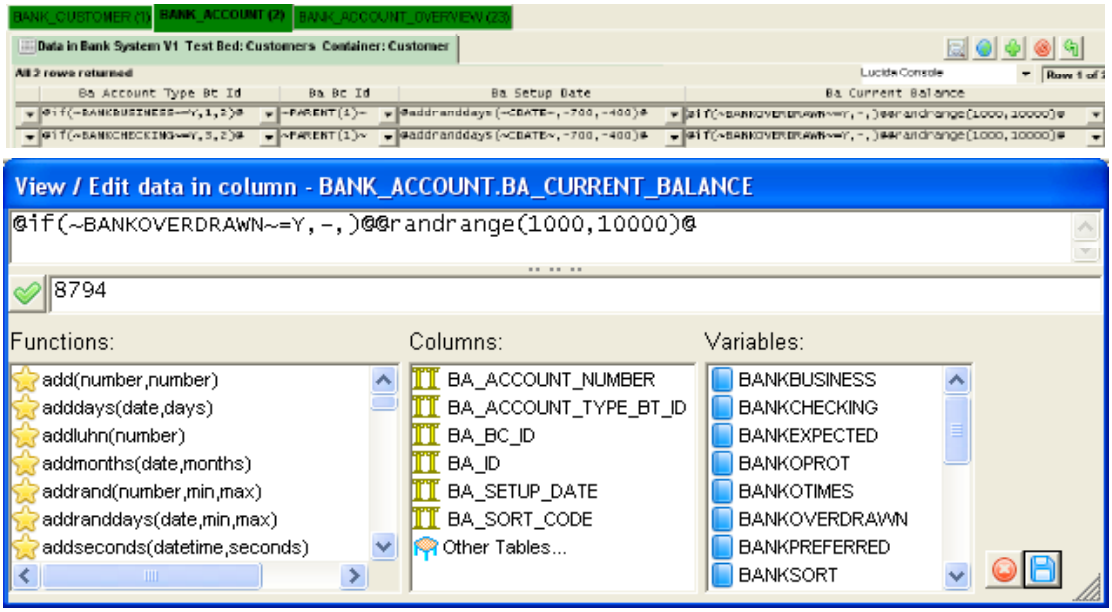


Figure 11 –In Datamaker, the column BA_CURRENT_BALANCE will become negative if BANKOVERDRAWN is set to Y.

4. Once you are happy with the definitions, generate a few test cases by hand and check that the results look OK in the application.

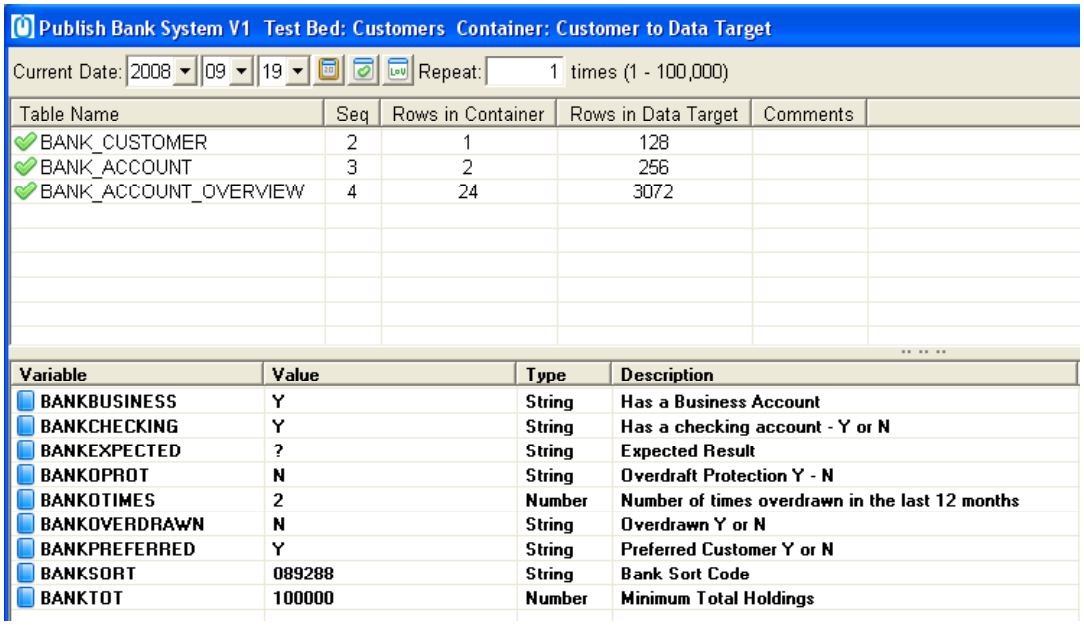


Figure 12 –A single test case data generation.

5. Use each test case input to drive each "publish". In this case, we have cleared down the data in the schema as part of the "publish".

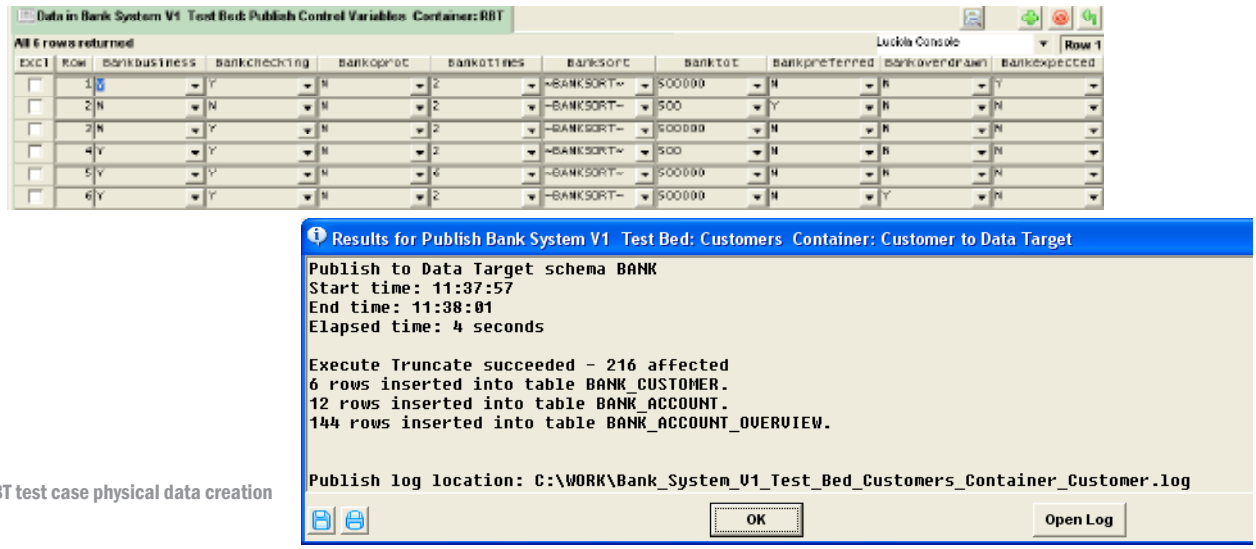


Figure 13 –The RBT test case physical data creation using Datamaker.

Expected Results

With products such as RBT, it is possible to derive the expected results from the test conditions; the cause-and-effect analysis will be able to produce an expected result. With Datamaker we have taken this result and included it in the data publishes. This allows the result to be easily checked against the actual result the program created. We used the column BC_NOTES to include a value EXPECTED=Y or EXPECTED=N. It is easy to then run a query to compare expected against actual result.

If no column is available in any of the tables, you can use other techniques such as updatable views, triggers etc. to check these results.

With most test case design algorithms, the expected results are not generated, which means that they have to be calculated by hand: A very strong reason to try and reduce the number of test cases!

Code Coverage Results For Overdraft Protection

After generating the data directly into the database using Datamaker for each of the test case combinations, the code was run and code coverage statistics gathered.

	Number of test cases	Expected results	100% code coverage
All Pairs	8	✗	✗
Jenny	8	✗	✗
All Combinations	128	✗	✓
RBT	6	✓	✓

Interestingly, the RBT methodology and tool outperformed all other methods. This is not that surprising, as the use of the internal code "nodes" allows the test cases to be significantly optimized. It would be possible to use triples or quadruple combinations of codes as part of the input rather than the paired code combinations used by jenny and all pairs. This would, however, increase the number of combinations.

Summary

- Increasing code coverage is the route to improved testing
- Validate requirements for ambiguity, use text parsing tools to validate clear text
- Identify cause-and-effect relationships
- Use advanced tools to generate minimum sets of test input combinations
- Do not use copies of production data to test with; they are too large and will not contain the specific test cases you need for maximum code coverage.
- Hacking existing data is error-prone and time-consuming
- Model existing data to generate and pivot values to create the perfect data for testing

Biography

In his career spanning over 20 years, Huw Price has been the lead technical architect for several US and European software companies. Specializing in test automation tools, he has launched numerous innovative products which have re-cast the testing model used in the software industry. As well as launching new products, he has also as entrepreneur built up three software companies, selling their technology to larger, less agile competitors.

Huw has provided high-level architecture design support to multinational banks, major utility vendors and health care providers. A strong believer in balancing pragmatism with a visionary approach, he has been able to rapidly bring new products on the market while maintaining strong quality control.

Huw's newest venture, Grid-Tools, has quickly redefined how large organizations need to approach their testing strategy. Grid-Tools has introduced a strong data-centric approach to testing, launching new concepts such as "Data Objects", "Data Inheritance" and "A Central Test Data Warehouse." Currently working with leading- edge testing companies such as Fiorano, Facilita and Empirix, Grid-Tools is building on the strategy of improving automation tools and associated structured methodologies.

Model-based Test Development and Automation

by Claus Gittinger

© iStockphoto

Today more than ever, corporations are faced with the task to develop increasingly complex products over shorter time periods while maintaining consistently high quality standards. This represents a particular challenge for the area of quality assurance. The trend to shorten project cycles increases the importance of comprehensive, efficient test development and automation.

In this context, model-based test development technology provides another approach to improve quality assurance processes while keeping to ambitious schedules. Here, test procedures are now modeled and no longer programmed. The test procedures can then be directly executed and analyzed in the form of a model. This universally intuitive presentation allows a larger audience to participate in the test development and analysis, including people with limited or no programming skills. Working with reusable building blocks makes test development more effective and efficient while reducing maintenance requirements for existing test procedures.

expecco is a model based blackbox-testing tool, where the interaction with the System-Under-Test (SUT) is completely modeled using activity diagrams.

End-to-end model based on UML activity diagram

What makes expecco special is that it is based on an end-to-end model driven approach derived from UML activity diagrams. According to experience, activity diagrams are intuitively understandable for all persons involved in the project, especially those with domain knowledge. The diagrams can be used for formal test documentation and in addition, they are suitable for execution and automation. Past project experience also shows that a high level of reusability is already achievable within an extremely short period of time, because the same methodology is applied from the abstract test case description down to the lowest level of building blocks for interfaces and protocols. expecco also offers building block libraries for a broad range of domains. As an added benefit, project specific libraries can also be created by the user. This enables a test team to work with a fixed set of building blocks and to combine those blocks into test scenarios.

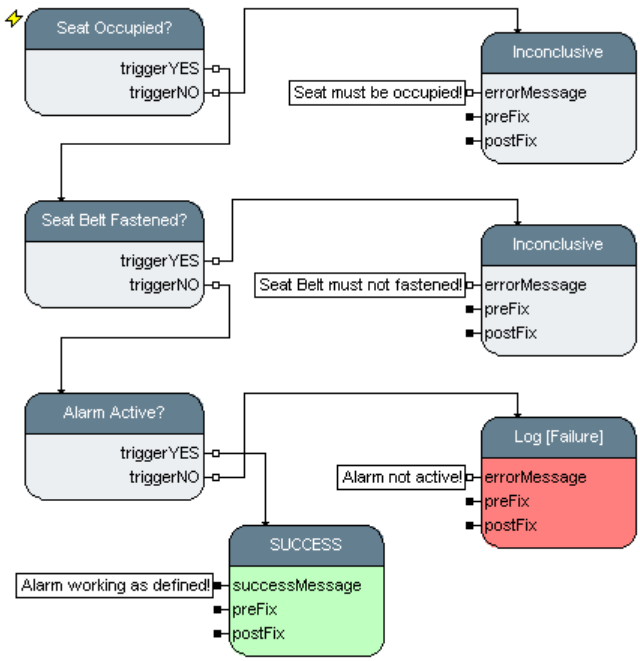


Figure 1 - Example of an automotive test case scenario in expecco

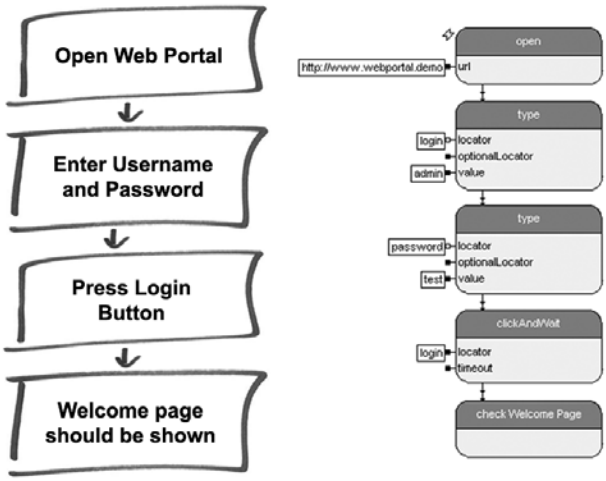


Figure 2: Test case scenario as a freehand sketch and as an executable activity diagram

Figure 2 shows a simple example of an executable activity diagram. The drawing on the left shows the process steps of a login process in a web portal. Such diagrams are developed very early in the project cycle during the analysis and design phase. It is a significant benefit that these drafted documents are not only used for documentation, but that they can be utilized directly as formal, executable and automatable test specifications. This is particularly advantageous when the internal details of the processes (interfaces, protocols, parameter etc) have not been defined yet or if they are variable.

The individual steps that are included in a diagram can in turn be described through additional sub-diagrams. Actions can be defined once and then also be reused in other diagrams. Using a drag & drop function, the login process in figure 2 can easily be inserted into more complex test case scenarios in the form of a building block.

In addition to these so called "compound blocks", which are defined through an activity-network, there also exist so-called "elementary building blocks". These elementary blocks can either utilize an existing service such as a SOAP, XML or DLL call, perform some database operation, or call a program function defined by a scripting language (such as JavaScript) or in an existing program.

The expecco execution engine allows for changes in an activity diagram or scripted elementary block to be applied at any time without a need for slow regenerations or recompilations. This is even possible while a test is being executed, making expecco also a perfect tool for explorative, agile test development.

Control and Automatic Parallelism through Data Flow

The synchronization of individual activities can take place through control flows as well as through data flows. In the case of data flow control, the synchronization takes place via input and output pins which control the transmission of data and objects. As soon as data appears at an input pin, the respective activity will be activated, which itself may generate data that will be transmitted to other process steps.

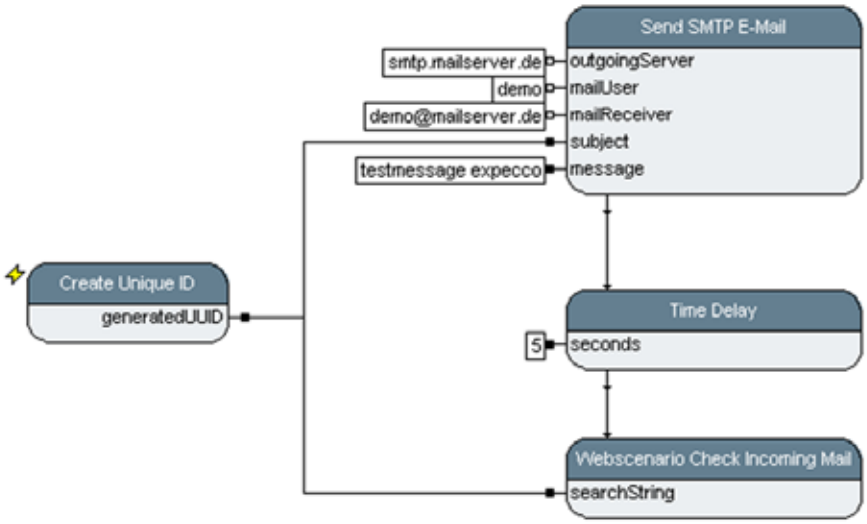


Figure 3: Network consisting of data and control flows

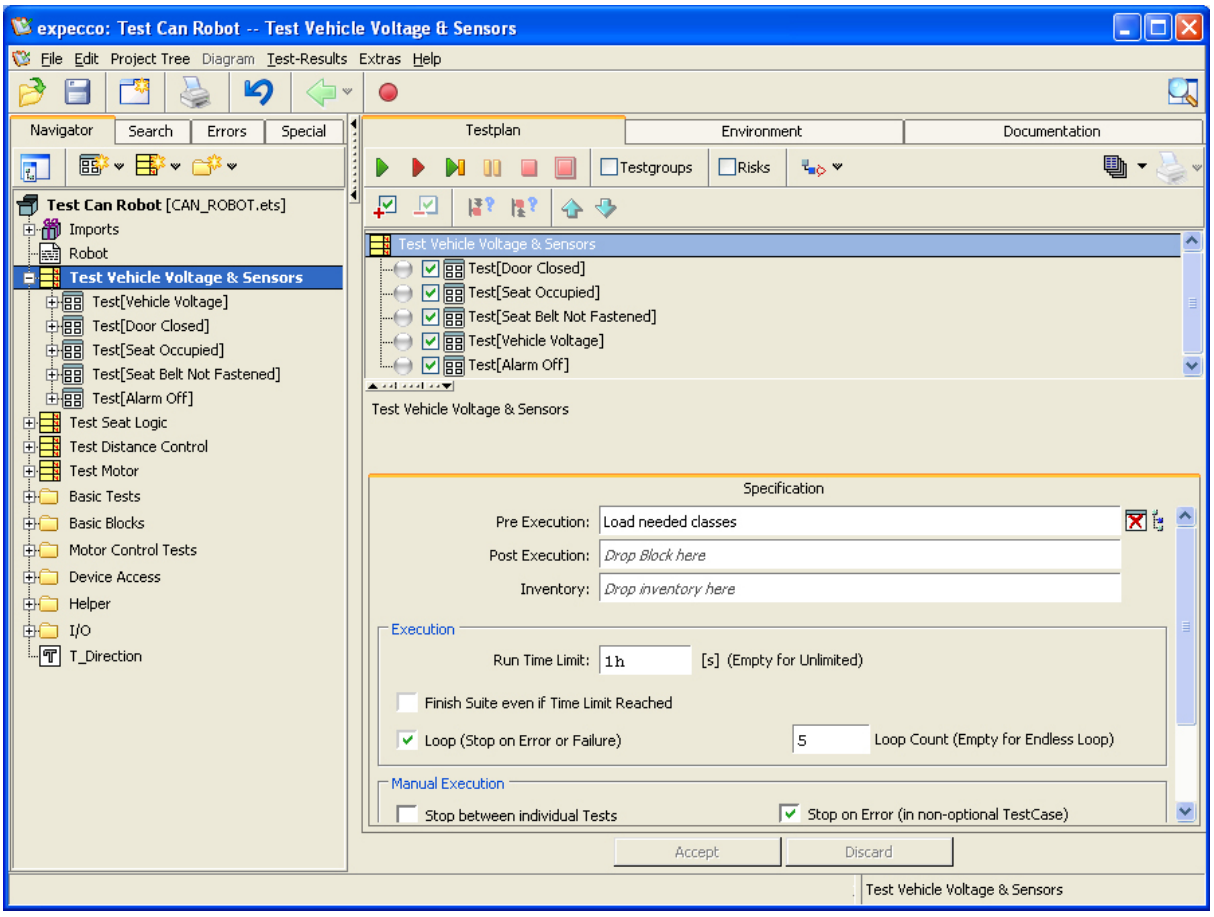


Figure 4: Test cases in a test plan. Ready to execute.

Figure 3 demonstrates data and control flows for the "Create Unique ID" building block which generates a unique UUID. With this data flow it activates the "Send SMTP E-Mail" step. The "Time Delay" step is initiated via a control flow, (a so-called trigger input), when the mail activity finishes. Finally, the lower block is activated, when its trigger input fires, even though the ID data has already arrived at its "searchString" input.

Even the producer-consumer problem, (which is a classic in the information technology arena when looking at the issue of process synchronization), is easily solved in the model by using configurable queues which ensure a defined producer-consumer behavior. Therefore, individual settings can either slow down the producer or can automatically start additional consumers on parallel threads. Error-prone synchronization methods, such as semaphores or exclusive program segments are not necessary.

From Activity Diagrams to Test Cases

In expecco, one activity diagram is associated to every test case. The activity has to stimulate the system under test and validate its responses. Thus, effectively simulating one aspect of the system under test's outside environment. This makes expecco both a test execution framework and a model executor. Figure 4 shows how multiple test cases are further grouped into a test plan.

Tracking Errors with Activity Diagrams

Inputs for the system to be tested are generated and conditions as well as responses are checked to verify a requirement or a defined behavior. Errors can be recognized and reported either explicitly via relevant building blocks, or implicitly through expected/actual status comparisons with default values.

After a test run, the execution results can be visualized via an extensive protocol. It documents all activities with time stamps, data flows, and exception information as well as explicitly generated messages. Furthermore, a configurable test report can be created either in HTML or in PDF format (Fig. 5).

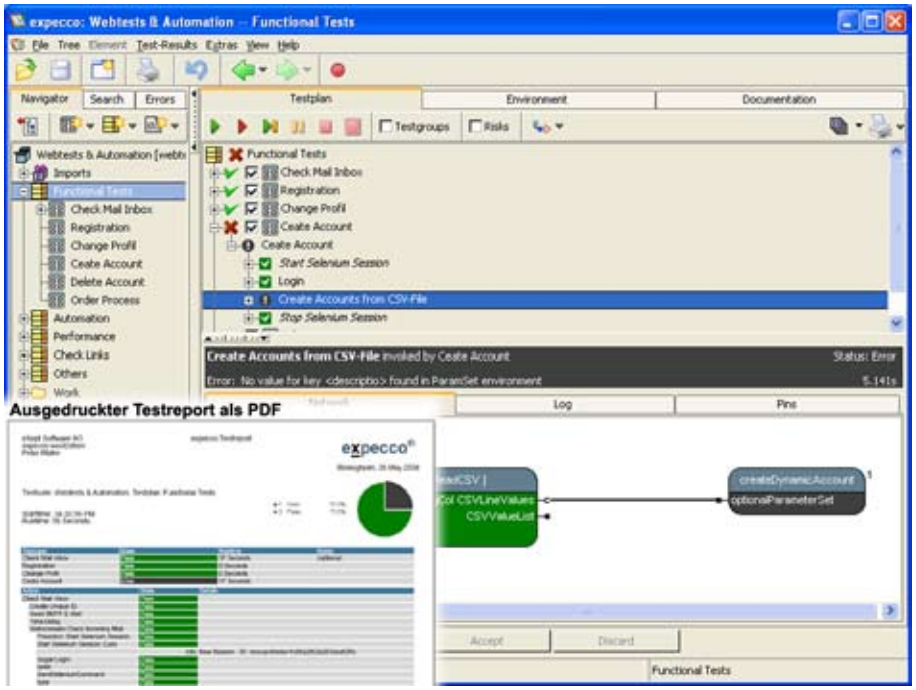


Figure 5: Extensive analysis options and reports after a test run.

Extensions for specific areas of application

expecco offers extensions for specific application areas, such as the testing of user interfaces like internet browsers and GUIs, or industry specific devices and protocols. One such example is the integration of Selenium; a web test framework for capture & replay. Interactions with a website can be recorded and transformed into activity diagrams which are then ready for further processing. In the area of automobile electronics, defined system functions are checked via CAN or MOST-Bus. Most recently, an extension has become available which enables expecco to automate CANoe from Vector, the development and testing tool for CAN, LIN, MOST, FlexRay and J1587.

Example

Here is a practical example taken from the area of web testing. It illustrates how a recorded web session can be captured and further processed as an activity diagram. This involves the recording of a login process as well as the completion of a web form which is subsequently parameterized via a CSV file.

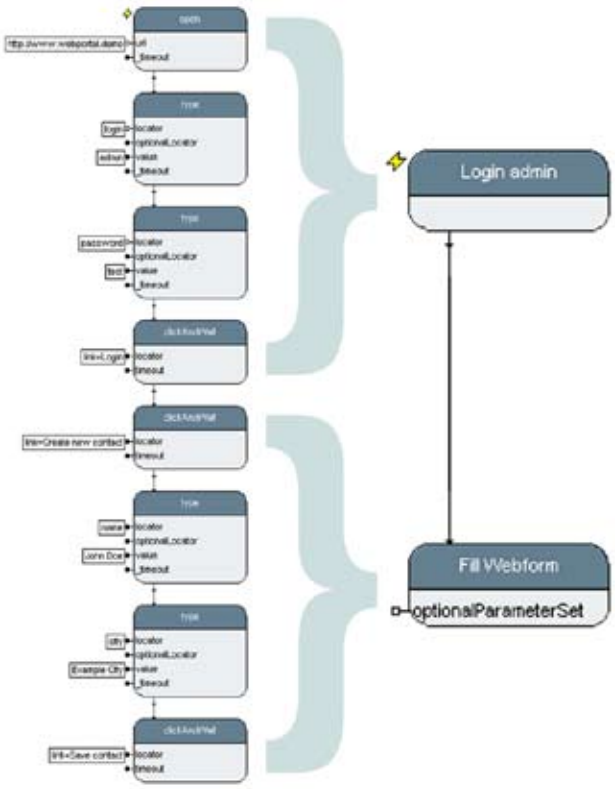


Figure 6: Logically connected process steps can be combined.

Figure 6 on the left shows an activity diagram as created by recording a web session. Each interaction with the website is represented by one step in the diagram. Logically connected steps can be refactored into new building blocks and can therefore be reused easily.

The first four steps on the left represent the login process and are combined into the new building block “Login admin”. The next five steps on the left include the filling out of a web form. These have been combined into a special web parameter building block called “Fill Webform”. This building block type enables the form inputs to be externally controlled via a set of dataflow parameters. A CSV file containing the input values of the recording is also automatically created during this process.

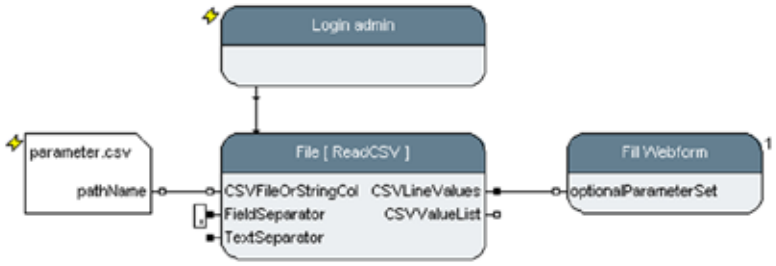


Figure 7: The completed test case scenario as an executable activity diagram

To complete our test case scenario, we drag the “File [ReadCSV]” building block and the automatically created CSV file into our network and connect the input and output pins as illustrated in figure 7. The CSV file can now be further edited and populated with additional values. For each line in the CSV file, the “Fill Webform” building block is then called. Alternatively, the parameter set could also be made available from the expecco internal variable environment, via an SQL building block from a database or even generated programmatically by another block.

Automation of Tests and Recurring Tasks

In addition to the execution of tests, it is also possible to graphically model or automate other recurring tasks with expecco. Practical examples of these features include installation tasks, verification of a ranking on the web, monitoring of the availability of servers or the completion of web forms.

expecco is being utilized successfully in a variety of industries such as telecommunication, automotive, engineering, software and IT for system and integration tests as well as for general automation tasks.

Companies benefit from the high degree of reusability and flexibility during the test development and are able to significantly minimize their expenses. Experience from a broad range of projects shows that the return on investment can be achieved within a few months. expecco and the model-based test development technology add new power to quality assurance.

Biography

Claus Gittinger’s professional background has been in software architecture for over 25 years with specialization on object oriented software. He is the creator of the Smalltalk/X programming system.

As a co-founder of eXept Software AG, Claus Gittinger has been defining and supervising project development and testing processes together with and on behalf of various customers. Past and current projects involve telecommunication, security, banking, factory automation and automotive processes.

Claus Gittinger is responsible for strategic product development and the continued development of eXept’s expecco quality management and test automation tool.

He also teaches computer science classes at the Hochschule der Medien (HdM) in Stuttgart on a part-time basis.

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

Test Process Improvement Manifesto

by Erik van Veenendaal

This column is a sort of follow up from my last column. Again I will be discussing one of my favorite topics, "Improving the Testing Process". Preparing for a keynote a this years' EuroSTAR makes one rethink the added value of test process improvement and models like the TMMi. There are many who claim great results by doing process improvement (either software process improvement or test process improvement). There are also many who claim that this is all a waste of time and everything should be done in an "agile way" – whatever that means.

Of course everyone knows or at least has heard about the agile manifesto. At EuroSTAR there was an attempt to create a testing manifesto. If manifesto's are a popular thing these days, why not create one for test process improvement manifesto? Basically in the test process improvement manifesto I have tried to define what makes process improvement work and what not. These are my recommendations for a successful application of models like the TMMi based on many years of practical experiences in various industries regarding process improvement.

Test Process Improvement manifesto

- **Flexibility** over Detailed Processes
- **Best Practices** over Templates
- **Deployment orientation** over Process orientation
- **Reviews** over Quality Assurance (departments)
- **Business driven** over Model driven

Flexibility over Detailed Processes

In general, having defined processes supports an organization. Only something that is defined can be improvement. It guides new engineers and acts like corporate memory. However building too rigorous processes takes away people values. You want good testers that have the skills to act based on the context of a problem and perceive testing to be a challenging job.. Supporting processes are needed but using the processes should give enough flexibility and freedom to testers to allow them to think for themselves and find the best way forwards. You only need "just enough process".

Best Practices over Templates

Templates are great but even better is to provide examples of how they should be used. What provides more support; a test plan template or three test plan best practices? I guess almost everyone working in practice will choose the latter. When doing test process improvement focus on getting a best practices library set up as soon as possible in stead of overspending on defining templates. Don't worry whether the best practices are the best in the industry. They are the best in your organization and if something better comes along replace them. This is what supports our testing and makes process improvement work.

Deployment orientation over Process orientation

Building process is easy, we have already done this so many times and there are numerous examples to be found. However, getting them deployed and thereby chang-

ing someone's behavior is the hard part. Process improvement is all about change management. I have seen test improvement plans that focus almost entirely on defining the testing processes. In successful improvement projects at least 70% of the improvement effort is spent on deployment – "getting the job done". Defining the processes is the easy part and should only account for a small percentage of the effort and focus.

Reviews over Quality Assurance (departments)
Communication and providing feedback are essential to project success. It is exactly this what peer reviews, if applied well, do. In principle also quality assurance officers evaluate documents and provide feedback to engineers. However, once to often I have experiences that quality officers, sorry no offence to those who do a good job !!, are too far away from the testing profession. Their feedback then focuses on conformance to templates and defined processes. Little added value to most projects, I believe. I have also experienced organizations where every test plan is peer reviews by one or two peer test manager giving feedback on the approach and content of the test plan. This is what we want, real feedback to we can use.

Business driven over Model driven
Whatever you do, make sure you know why you are doing it. What is the business problem you are trying to address? What is the test policy supported by management? Just trying to get to TMMi level 2 or 3 without understanding the business context will always fail in the short or long term. When addressing a certain practice from an improvement model, there are most often many different to comply. The business problem (poor product quality, long test execution lead time, costs, etc) will tell you which one to choose. Almost continuously review your process improvement against the business drivers and test policy.

When doing test improvement take the above mentioned starting points, values or whatever you like to call them into account. I'm sure when using them in the right way you will get better results. Some have called the manifesto "lean and mean test process improvement". An interesting thought, perhaps something for a next column ……

Doing Automation the Smart Way!

by Adrian O'Leary

In today's competitive age, the high degree of customer interaction and increased intricacy of systems magnifies the risk of failure associated with software applications. With a large number of applications going online through the internet, applications that were once used by business users have now been exposed to large end-user communities. Consequently, any bug in the application directly impacts the brand image and reputation of your organization. Studies show that finding and fixing defects during test execution can cost 50 times more than during the early requirements phase, and as much as 200 times more if left undetected until after production. Hence, it is crucial for organizations to adhere to structured testing processes and thereby deliver higher quality systems in less time and with fewer resources.

Consequently, testing assumes a critical role in application development and maintenance and application test engineers need to cope with issues such as compressed project timelines, frequent application changes, lack of well defined business requirements, amplified security concerns and unpredictable user loads. With every feature loaded to an application, it becomes imperative to run the entire gamut of testing again. Executing these manually sometimes could be mundane and time consuming. Organizations look at test automation as the cure-all for their problems in improving testing cost and effort. Availability of tools and frameworks for automation help testing teams construct and maintain reusable automation scripts enabling teams to test faster. But while addressing many of these problems; it should be known that automation in itself cannot unravel all of them.

In order to realize noteworthy returns, the scripts created need to be reusable and robust for all users to execute them. This can be ensured only if the automation objectives are defined in advance and the entire automation approach is well planned. Automation requires a significant up-front investment for building the automation test suite. The QA team should be equipped to identify automation opportunities, evaluate and select the right automation tool, adopt industry best practices in scripting and maintenance and thereby increase the return on automation investment.

Not everything is meant for Automation

Organizations try and automate all their software testing and have them executed as quicker as possible. However, it is always good to keep in mind that the investment on automation is very high and sometimes the cost-benefit of automating everything is not always advisable. Thus, for successful test automation we need to investigate in the right set of tests that can be automated effectively and efficiently.

C2Auto
C2Auto helps in identifying the ideal Test Cases for Automation

To perform this, you need to first do an evaluation based on parameters such as frequency of execution, reusability of scripts, resource availability and dependency. There are few decision making tools in the market that help you identify the tests that can be automated effectively and efficiently. For example Cognizant's "**C2Auto**" follows the scientific methodology to help the testers decide on the tests that can be automated cost-effectively.

Calculate Return on Automation Investment

Automation Testing involves higher initial investment in terms of tool procurements, training, scripting etc. However the costs associated with test execution reduces over a period of time. Performing a return on investment (ROI) analysis on each automation project will help determine the types of automation that can be done for the project. This can also help us arrive at the tools that may be required for the successful completion of the automation project. Not only does ROI serve as a justification for the effort, but also is a necessary piece for the planning of an automation project.

From our experience, we have realized that projects that do not perform ROI calculations upfront end up wasting time and effort in non-essential areas. There are structured ways that automation teams can adopt to arrive at this value. For example, Cognizant's "**ROI calculator**" can help you derive an accurate return on automation investment. The tool can compare short-term cost savings as well as the long-term gains for automated and manual testing in a project. It computes the savings from automation in terms of cost and effort, and thus provides high level statistical information for project management.

ROI Calculator

Transparency
- Provides key project viability estimates
- Provides accurate Return On Automation Investment
- ROI estimate split provided across automation activities
- Provides the information about break-even point

Portability
- Web based tool (used anytime-anywhere)
- Independent of automation tool used
- Option to export reports and graphs

Ease of use
- Dynamic calculation of ROI
- Provides comprehensive graphical & tabular reports on
  - ➤ Cumulative cost and
  - ➤ Cumulative effort

Smart Approach to Test Automation

While crafting an approach to test automation, you also need to consider factors that influence automation efficiency such as ease of maintenance, portability of scripts etc. The objective of Test automation is to increase application reliability, while reducing the time and cost of software quality programs during the test process.

Automation approach could be smart only when it supports test efforts involving automated test tools and incorporates a multi-stage process of how to introduce and utilize an automated test tool. The approach needs to cover analysis of requirements, automation test planning, and test design while also addressing test execution and test management. All these elements including tools and utilities can be integrated to accomplish end-to-end automation.

For example Cognizant's "**Automation Tool kit**" has the set of framework, tool evaluation approaches and utilities bundled together to ensure efficiency throughout the automation lifecycle. We have knitted together an entire gamut of tried and tested automation techniques to accomplish the following objectives:

- Ensure higher efficiency in automation and cutting schedules
- Ensure higher coverage
- Enhance reusability
- Ensure ease of maintenance and portability of scripts

Approach to Tool Evaluation and Selection

Selecting the right tool for automation has been a challenge and this requires tool usage experience which helps reduce the complexity, while ensuring adoptability and
fitment. Prior knowledge on the technologies adopted for development is required to identify the list of tools that will best serve automation requirements. You should demo or trail to pilot some of these tools real-time to understand the suitability and effectiveness.

While deciding on the test tool, the schedule is reviewed to ensure sufficient time is available for test tool setup and development of the requirements hierarchy. It is a good practice to map potential test tools and utilities to test requirements. It is also suggested to confirm test tool compatibility with the application and the environment. In case of any issues it is suggested to have a few workaround solutions investigated to address incompatibility issues that may surface.

Arriving at the Best-fit Automation Framework

Drafting a framework for automation is important to ensure maximum reusability resulting in higher efficiency. It is suggested that Test Automation be integrated into a centralized function that owns all the reusable components. This helps remove all redundancy in the system that may arise out of obsolete solutions and components.

At Cognizant we have built an exhaustive framework for automation called "CRAFT"

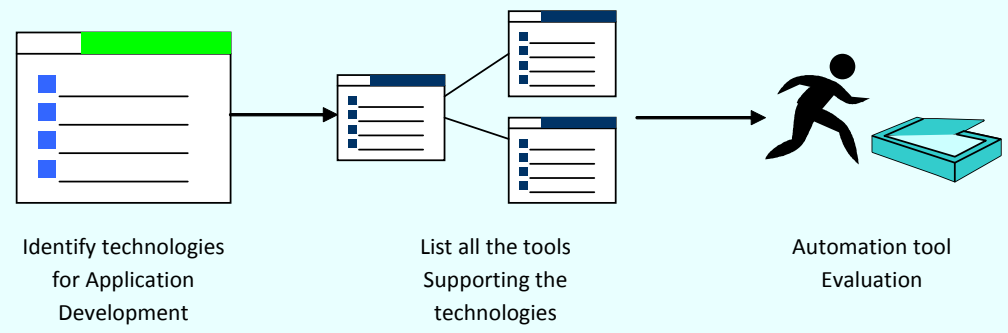
"CRAFT-Cognizant Reusable Automation Framework for Testing", is a robust end-to-end Automation framework. CRAFT defines the method for scripting of business functionalities as reusable libraries that are repetitive among test cases. This simplifies automated execution of large number of test cases by using a centralized engine (i.e., Driver Script) that invokes the relevant libraries as per test case requirements. Functionality based scripting paves way for reduced number of test scripts with maximum reusability and minimum redundancy. The driver script has the logic for calling different reusable actions in a sequence based on functionalities that are to be tested. The framework also has a Database Abstraction Layer, which has business scenario test data and provides input to the driver script for individual test script flows.

CRAFT
CRAFT defines the method for scripting of business functionalities as reusable libraries that are repetitive among test cases

CRAFT 2.0
CRAFT 2.0 is a tool which streamlines the test execution activity during test automation; it dynamically executes the test cases in multiple machines in a distributed environment

Innovations that compliment Automation

There are many industry standard tools widely adopted by QA teams for automation. But the issue is that these tools predominantly address one or only few areas of automation and requires to be customized to cater to all automation requirements. At Cognizant, we have built a full suite of innovative solutions to compliment the automation tools and process and to increase automation efficiency. Some of the tools and utilities developed by our team are displayed below



DataXpress

Data plays a vital role in testing and data generation has been the greatest challenge for most of the QA teams. To address this issue, Cognizant has developed an automated test generation tool named "Data Xpress" that helps QA teams to streamline the test data preparation activity through automatic generation of test data.

AHEAD & WIN2PRO
QA teams often face the challenge of migrating from a tool to another or upgrading the current tool. At Cognizant we have built solutions to address these types of challenges. For example "AHEAD" is a tool built to support bulk uploads QTP scripts, attachments and folder structure to Quality Center. This significantly saves time and effort associated with manually porting these from one to another. Similarly "Win2Pro" performs quick migration of WinRunner scripts to QTP with minimum effort and time. The tool supports
application user interface like Java, VB, Web and ActiveX. We have witnessed this tool to reduce manual conversion effort by almost 80%.

LiBex
Automation efficiency can be enhanced only with increased reusability of scripts, functions and components. To facilitate this we have "LiBex" that functions as a search engine to retrieve functions present in one or more libraries. It also facilitates easy download of functions thereby facilitating increased reusability.

ORGen
Conversion of object repositories into formats understandable by the system is often a challenge to most QA teams. To address this challenge, Cognizant has developed "ORGen" that will help eliminate all manual efforts by automating creation and conversion of object repositories from one format to other.

WS Test Professional
Testing web-based applications have always been a challenge because of the absence of user interface. WS Test Professional is developed to address this issue thereby making testing of web services easy.

Customize and use Open Source tools and Frameworks

From our experience we have realized that open source tools in many instances could be customized for use thereby saving the cost associated with license procurement. Also open source offers greater flexibility and provides solutions for testing areas where industry standard tools are not available. For example, tools for testing Web applications or automating testing for Web-applications are still emerging. At Cognizant, we have customized few of the tools and frameworks available in open-source to provide solution for Web-testing and Web-automation. For example "WATIR Framework" is an open source functional testing framework customized to test any Web application built on ASP, .NET, J2EE or PHP. Similarly "Selenium Test Manager" customized from Selenium, an open source tool that aids Web test automation. Defect management open source tool "Bugzilla" is customized and integrated with test management tools to take care of the defect management portion, helping to reduce procurement cost of licences.

Conclusion

QA organizations have adopted Test Automation as a viable option to address QA challenges related to reducing cost and ensuring higher coverage. Test Automation in today's competitive environment is much needed than desired; to keep operating efficiently and considerably cut down costs and efforts, without compromising on quality and security. However you must adopt a well planned and a structured approach to automation in order to ensure a higher return on investment. It is suggested that prior to opting for automation, QA teams need to perform an exhaustive automation assessment to identify the right set of automation methods, tools and techniques that will compliment their QA needs. Smart QA organizations leverage automation experience and expertise of testing service providers such as Cognizant for automation assessments, tool consulting, solution framing and implementation.

These organizations have thus been successful in garnering higher return on automation investment through the adoption of industry best practices, streamlined processes, and reusable automation frameworks - The secret behind doing automation the smart way.

Biography

Adrian works at Cognizant Technology Solutions as Director of Testing Centers of Excellence for North America. Adrian has more than 16 years of experience in the Software QA and QC arenas. Adrian is very active in the QA Arena and sits on the Board of Directors for HP Software's Global User Community called and Vivit

Stephan Goericke

Interview Andreas Golze and Stephan Goericke

Andreas Golze

Andreas Golze,
Global Practice Director Hewlett Packard GmbH

Stephan Goericke,
Director International Software Quality Institute GmbH

TE: Mr Golze, Hewlett-Packard has decided to invest in quality assurance through personnel qualification. What do you expect to achieve by that?

Golze: The experience of the last 15 years has shown that we need a predictable quality in consulting services to be truly successful in the market for quality assurance and governance. When I employ software developers nowadays, I won't take someone who has been an exceptionally good hacker at school, but someone who has experienced a sound professional training.

TE: Mr Golze, you have decided to use the ISTQB-Tester-Standard and also the IREB- Requirements Engineering-Standard. Do you deliberately opt for manufacturer independent standards?

Golze: HP generally tries to assimilate and actively contribute to industry standards. At the end of the day this guarantees that no manufacturer can define a standard and then simply lays his hands on his employees and declares that they are competent in applying it. Therefore we have an independent controlling organisation which helps us to keep up our quality standards. We invite independent external assessors to check on us, so we can prove that we are doing it right. We have audits in all kinds of different areas. We use QAMP now, so that our staff has quality standards that can be confirmed independently.

TE: The idea of quality assurance or respectively personnel qualification is not new to Hewlett Packard. HP is one of the initiators of the German Association for Software Quality and Further Education (ASQF), which set up the International Software Quality Institute (iSQI). Is this why you decided to choose iSQI as your strategic partner?

Golze: On the one hand, iSQI has been there from day one and therefore possesses the required competence. It is not a niche player, but rather a proper certification body that is recognized by the market and which we can mention in connection with our name with a clear conscience. On the other hand, there are many people already working with iSQI, among them other global players like T-Systems or SAP. This means we are well advised to work with this certification provider, who has the competence and above all the capacity and capability of being available for us as certification partner on a global scale.

TE: Mr Goericke, Hewlett Packard uses QAMP for the certification of its employees. You have developed this standard. What characterizes QAMP?

Goericke: QAMP stands for Quality Assurance Management Professional and is a modular certification standard which is tailored according to the needs and work environment of employee and employer. On this basis it is possible to gradually gain professional skills on a very high level without having to resort to a costly second degree or lengthy professional trainings. Companies which are desperately seeking skilled personnel can now search for employees with potential. By using specific further education they then introduce them very quickly and very flexibly to precise requirements. Or they can let the present staff grow with new challenges. Mr Golze already mentioned it: There are several global players like Hewlett Packard or SQS [Software Quality Systems AG, editor's note], who have already started their internal further training on quality assurance in accordance to the QAMP scheme. But there are also smaller companies which take their chance to further educate their staff quickly, up-to-date, affordable and in accordance with the work processes, so that they stay fit for future challenges of the market. Furthermore, a lot of employees are very grateful to at last receive a certificate – and thus recognition – not only for their theoretical knowledge, but also for their long-time practical experience and lifelong learning.

TE: Mr Goericke, you mentioned a modular certificate. What do you mean by that?

Goericke: The QAMP scheme is based on four modules. Three of those are training modules. The fourth proves practical experience. This sys-

tem makes it possible to start your training at one place and then continue with it in another place at another time. As the relocation of employees for reasons based on company strategy or according to customers' request becomes more and more common in the IT sector, it is exactly this flexibility that many consider the key advantage of the scheme, as it enables them to adjust their further education to their work environment. After the successful completion of each module you receive a certificate.

TE: Which certificates exactly are you talking about?

Goericke: The basis for a QAMP applicant is standardized knowledge in the fields of Requirements Engineering and Software-Testing – QAMP modules one and two. This is why we consider it sensible to stick to internationally recognized certificates such as IREB Certified Professional for Requirements Engineering and ISTQB Certified Tester. With these qualifications, basic knowledge will be imparted and the future quality assurance manager will develop the necessary understanding of the entire software development process. Once this knowledge is established, one takes the first step towards specialization. With QAMP this should happen in the third module. Here the applicant should chose an exam on another domain, which does not only do justice to the former or future work environment of the QAMP applicant, but at the same time completes or demonstrates the knowledge he or she has acquired there.

TE: Please name a few examples!

Goericke: There is a wide range of opportunities for specialization. Consider for example Software Test-Management, Configuration-, Innovation- or Project-Management. A lot of people also specialize in Secure Software Engineering or Software Architecture. Domain specific certificates for the medical or financial sector are also possible. In the end this third module enables the highest possible adjustment of further education without leaving the framework of a standardized and independent qualification and certification process.

TE: How does this model differ from other already existing modular certificates?

Goericke: I think by looking at two factors you can clarify the difference to other certificates: First, there is experience, which is included in QAMP by means of the fourth module. You have to prove two years of practical experience in the field of software quality assurance to receive the QAMP certificate. For the certified person it is becoming more and more important these days to be able to produce a measurable equivalent value, a proof of his or her practical experience. What else can older employees put in the balance when confronted with younger competitors? The keyword "experience" alone will not do the job; you need an internationally recognized confirmation, a stamp of quality. That's exactly what QAMP is. The second factor next to the inclusion of experience is the fact that lifelong learning is guaranteed.

TE: But how can you guarantee someone's lifelong learning?

Goericke: The QAMP certificate is only valid for one calendar year. To extend the validity of the certification one has to prove one's practical experience anew. I think Mr Golze agrees with me that this alone is the right direction to meet the high requirements of the fast moving IT sector.

Golze: I think it is completely reasonable that QAMP requires the proof of practical experience. We don't gain anything by employing great theoreticians who have no understanding of practice. You have to apply what you have learned, this way it will remain available to you for much longer.

TE: Mr Golze, Mr Goericke, where do you see your business venture QAMP in 3 years time?

Golze: I believe that if we achieve with QAMP what we have already achieved with ITIL, then QAMP will become the quasi-standard that people in quality assurance, testing and governance will be measured against. I am not saying that QAMP will be the only standard, but every new one will have to prove that it is at least as good. Because QAMP combines the three different areas to a logical basis. The beauty of it is: We will have our staff re-certified every year. This means that we will not have the same effect as with other certificates: once you are certified, you forget all about it for the rest of your life. A QAMP certificate holder continuously looks into the subject, either in his or her active professional life via seminars or any other forms of publications. It is not a theoretical certificate in the sense of "lovely to have learned that at one point". The certificate holders are always on top of things, developing themselves further. I would imagine this is great for the employees, too, for they can thereby increase their market value. And I assume that in three years time we will see in the job advertisements that QAMP certificate holders are sought-after in the areas of quality assurance and quality management.

Goericke: If it came to that, this would be pleasing. It could happen sooner than that. We can see that there is great interest in the QAMP certificate abroad, too, for example in India. I am convinced that further education in the future will follow the basic idea of QAMP – the combination of theory and practice.

Risk investment Test Automation?

by Konrad Schlude

Abstract:
Test automation can be regarded as a risk investment. Some authors report on a reduction in manual test effort of 80%. There is, however, the number of about 63% of failed test automation projects. On one hand, there is a high potential gain, on the other hand there is a high probability of loss of investment. In order to find out why there is such a gap, we will consider two test automation projects within one company. Whereas one of these projects is considered as a success story, the other was just an investment of several man years without any contribution to the overall software development project. As these examples will show, the success depends on many factors, especially on a good setup.

Obviously, tools are needed for test automation. And every producer of such a tool will argue that his tool is a good one. The question of which tool to use is an important one in test automation, since every tool has its advantages and disadvantages. However, the tool question is not the only question in test automation. It is just one of the reasons why many automation projects fail. Brian LeSuer puts the number of failed automation projects at 63%.

To illustrate some reasons why test automation projects can fail, I would like to give two examples of software development projects. In both projects, test automation was initialized. In one these projects, test automation was and still is regarded as an success, since and there was an important contribution to the software development over several years, In the other project, test automation was nothing but an investment of several man years without any remarkable outcome. Interestingly, both projects were in the same company, in the same building, and on the same floor.

Let's call these projects Project A and Project B. Although there are many similarities, there are also many differences. In Project A, test automation emerged from the test team. The test team developed a strategy of what to automate, so knowledge about the application was available for test automation. In Project B, an external automation team was brought into the software development project, and this automation team acted as a closed group, seperated from the other teams. They did something, but outside the automation team nobody knew what was going on.

In Project A, the test team checked every run of the test robot. With their knowledge, the team members were able to analyze every outcome of every run. If a modification of a script was necessary, a corresponding order of what to modify was generated. On the other hand, when a bug was detected, it was reported immediately to the development team with the request that the bug should be fixed as soon as possible. And the developers fixed the bug as they were told to do. So test automation kept running and kept finding bugs.

In Project B, the automation team was not able to do anything similar, since it was separated from the knowledge of the other teams. Failed test cases were just red marks in the report, and no further analysis could be made.

From the start, maintenance of the test automation was an important topic in Project A. There was a concept of how to react to changes of the application, i.e., the scripts were organized in a modular way. Every GUI element (e.g., button, link, entry field etc.) was identified by a variable stored in a single repository. Furthermore, the team members developed programming guidelines for generating scripts. With this, the amount of time needed for maintenance was quite small.

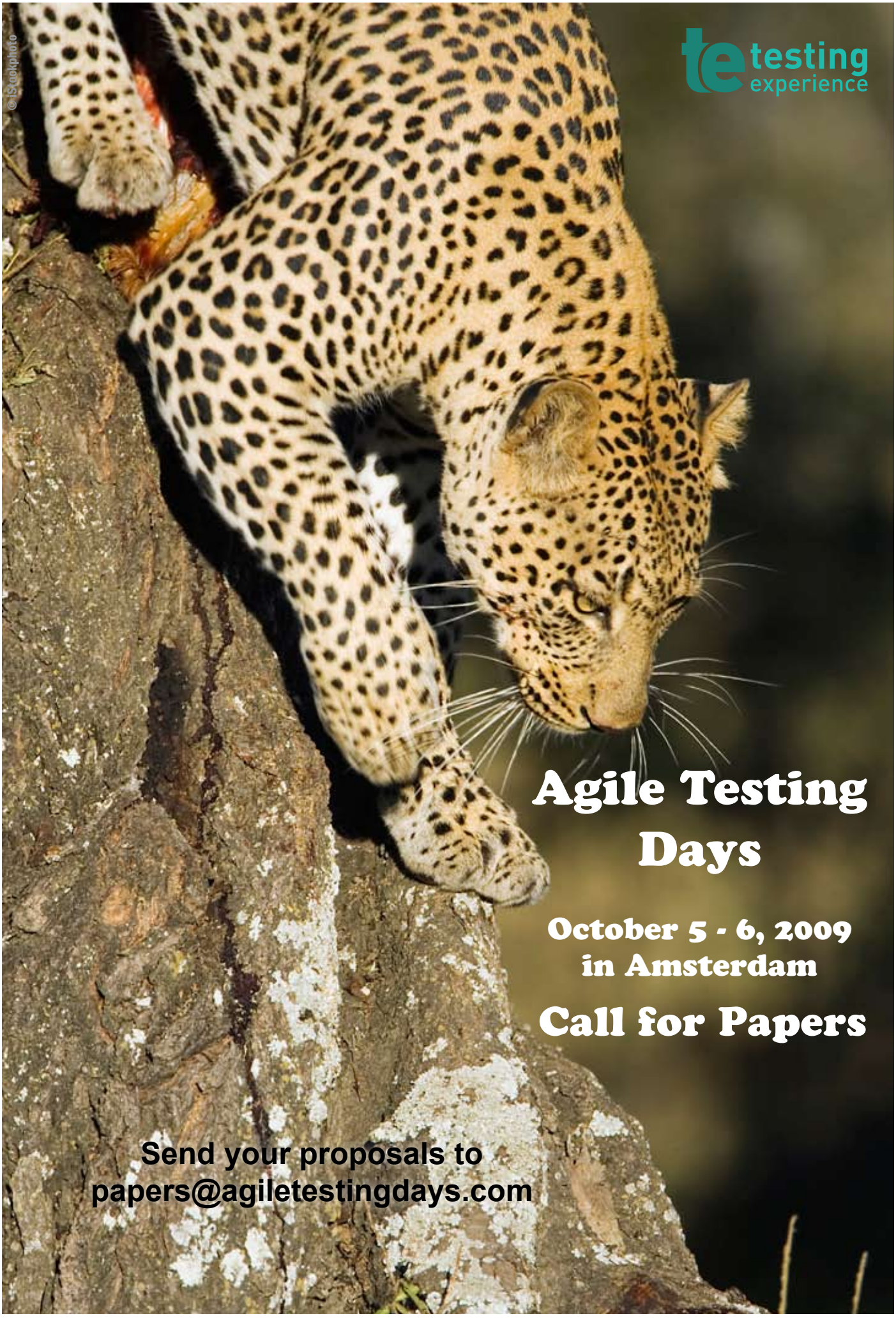
Again, Project B was different. The automation team just used the capture functionality of the tool without any idea of modularization of the scripts. In the beginning there was very fast progress. After a while, however, they were not able to react anymore to the changes in the application. It was the typical situation where a small change in the application (e.g. button was renamed) led to the modification of an enormous number of test cases. The automation team was busy all the time, but they could not follow the development progress. In order to solve special tasks in some test cases, the automation team programmed scripts. But since there was no concept behind these scripts (i.e., there were no programming guidelines), no one was able to understand the scripts after a short while.

It is not surprising that the automation within Project B failed. Virtually everything went wrong from the beginning. On the other hand, test automation in Project A also had a possibility of failure. For instance, the help from the developers was crucial for success, but there was a lot of goodwill, even though this had not been defined before. Since the personal communication within the triangle of test team, automation team and the developers was OK, every problem could be managed. One reason for the success in Project A was that the involved people brought their knowledge together.

This leads to the following conclusion. Test automation is not a risk investment; it is just a waste of time and money if you don't have a good setup. Just doing "something" with a tool is not a good idea. However, if you define the right frame and bring knowledge and people together, then you have a good chance to substantially improve software development.

[1] Brian LeSuer: Supporting Steps for Successful Test Automation Projects, SQGNE, 2004-05 Presentations

[2] The notion of programming guidelines might be surprising in this context. However, typically manual editing of the scripts is needed although powerful capture-replay tools are used. Within one automation project the amount of manual editing was 80%, and only 20% of the time were used for capturing. And since there is a lot of editing / programming, there should be programming guidelines


© iStockphoto
testing experience
Agile Testing Days
October 5 - 6, 2009
in Amsterdam
Call for Papers
Send your proposals to
papers@agiletestingdays.com

Improved time to market through automated software testing

by Dr Mike Bartley

I have experienced many business advantages through the automation of software testing.

- Reduced costs as automation reduces the manual effort and thus the cost of testing.
- Improved quality through both regular running of the automated tests and having more time for running the remaining manual tests.
- Improving employee motivation by reducing the amount of tedious manual testing.

A test team should always keep in mind their contribution to the business, and in this article I will concentrate on **improving time to market** through test automation. Automation can make the following key contributions to time to market improvements.

- **Getting test results sooner:** The results from an automated test suite should be available sooner than the manual equivalent.
- **Catching bugs earlier:** The earlier a bug is found, the quicker (and cheaper) it is to fix. The ability to regularly run an automated suite should ensure some bugs are found sooner.
- **Stabilising the software sooner:** The regular execution of an automated regression test suite should allow us to stabilise the software sooner.
- **Assessing the stability of the software:** One of the key questions is always "are we ready to ship?" and the test manager needs to be able to provide data to help with that decision. The automated test suite should provide a history of failure rates and can also be a means to generate structural coverage information.

However, although the business benefits are potentially large, automated testing needs to be approached with extreme caution due to the high risk of failure!

Automated testing tools fall into two main types: static and dynamic. Static analysis tools have made huge improvements since the early days of "lint" and can now find a range of complex issues: from memory leaks to race conditions; from naming conventions violations to uninitialised variables; from "island" code to dead-end. The tools occasionally report false errors, but these are reduced through improved analysis engines and improved "context" by reading in larger chunks of source code. Static test tools are reasonably easy to automate, because it usually just involves reading the source code into the tool. I recently automated static analysis across the complete source code base by making a few changes to "make" files in an automated build system. The challenge is then to prioritize the issues and get them fixed. In [1] VDC predict that the market for static analysis tools will overtake that for dynamic tools for embedded software testing in 2009.

Static testing is excellent at improving the overall quality of your code although its contribution to accelerating our time to market is not direct. Also, it is not sufficient as it does not test that the software implements the required functionality. For that most companies turn to dynamic testing . In [2] Kaner states that "automated testing tools" only actually automate some of the tasks required in dynamic testing: running tests and reporting results. The test engineer will initially need to design, create and document the tests, add them into the automation framework and get the tests running for the first time. There are tools that can automate some of these tasks (e.g. automatic test data generation based on source code analysis) or help with these tasks. For example, JUnit provides an excellent environment for writing new tests and tools for automating their execution. Finally, the automated tests will also need to be maintained, and it is here that attempts at test automation often falter. For example, early attempts at automating GUI testing produced tests which were too brittle and failed after minor changes in the interface. Such tools are now more robust and GUI testing is now an obvious choice for test automation.

One of the key decisions in accelerating time to market through test automation is deciding which tests to automate and which to leave "manual[3]". I have found the following guidelines useful over the years:

Complexity of the test environment: The more complex the test environment, the more maintenance the automated tests will require. For example, one recent project I was involved in required us to carefully measure performance improvements gained through use of the product being tested. This required the test environment to be set up very carefully (e.g. to ensure the machine had no background jobs such as virus checkers running), and we found it easier to do the setup manually before running a series of automated performance tests.

The level of testing: I usually find it easier to automate unit testing than system level testing, because the unit test environment is less complex. There are also a number of good testing tools (many open-source) to aid unit level test automation. User acceptance testing is often not a target for automation, because this is often a formal signoff procedure with the customer. However, the ability to automate the types of tests that users want to run is helpful to ensure that user acceptance testing is likely to be successful.

How often will the test need to be run? If a test is going to be run very infrequently then the ROI (return on investment) from automation may not be so obvious.

Ease of automating the pass/fail criteria: It is not always easy to automate detecting whether a test passes or fails, and the effort taken to set this up means the ROI on automating the test does not stack up.

Test stability and repeatability: For example, if the code under test is not stable, then the lack of stability and repeatability of the test means that the maintenance costs of the tests are too high to consider automation of these tests.

When choosing a test automation tool, your crucial consideration should be the target user community. A number of tools market themselves on their ease of use and simple scripting language, assuming that the users are not familiar with software coding. Other tools can link with standard programming languages, thus allowing common libraries of test code to be developed and re-used. In one automation project I observed that the developers were writing unit tests for automation, but found

[1] It depends on what tests are automated as to what sort of bugs are found. It is unlikely that every possible bug will be found via an automated test suite.
[2] There are formal mathematics-based static techniques for testing correct function, but these are not widely used.
[3] Even "manual" tests can benefit from some level of automation. For example, scripts to set up the environment before running through a few manual steps.

A Communit

100,000 Ce

Are you on th

www.is

*October 2008


y Worldwide!
rtifications*
e right track?
tqb.org
ISTQB
International Software
Testing Qualifications Board

the simple scripting language that came with the tool extremely frustrating. Although the automation ultimately helped time to market, it could have made a much greater contribution if the tool had better matched the skills of the user community. A few other important considerations in tool choice are:

- The ability to separate orthogonal concerns such as test setup, test environment, test data and test procedure. One test framework I worked on allowed us to use files and environment variables to define the test environment, test start-up and tear-down, test data and expected output. This easily facilitated reuse in the test environment.
- Portability of test between environments/platforms. For example, you may need to test your software on both Linux and Windows, with a wide variety of peripherals.
- The ease of adding tests to the test pool and temporarily banning tests (for example whilst a particular piece of source code is updated).
- Ability to integrate the automation with the software configuration management and build system.
- The cost of the tool is obviously a consideration, but I would caution against rating this factor too high. The cost of ownership of the tool will usually outweigh the initial purchase cost.

My last two automation projects have both been home-grown using scripts and make files. This gave us several advantages in terms of flexibility in our test generation, but we did observe a major drawback: employing external test development resource (through either outsourcing or contractors) meant that delivery of the tests was harder as they were required to understand our bespoke test system and we obviously couldn't find suppliers who had prior experience.

Once you have your tool, you will need to maintain the tool (as well as the automated tests). In my experience it is not always advisable to give this task to the test team, as they do not always have the appropriate knowledge and skills. So, for example, if you have bought in a tool, you should then consider allowing the IT department to maintain it. If you are building your own tool, then it might be best for the software development team to develop and maintain it. You will also need to integrate the tool into your development process. I have found that integrating the automation into the build process is the most valuable approach. This has meant that whenever we built the software we got an automated set of test results too.

You will inevitably be asked to measure the ROI for the tool. The cost of buying the tool and licensing it is easy to measure. It is also possible to measure the cost-saving in terms of time saved through automatically running tests vs. running them manually. However, you have to add in the cost of maintaining the tests. Also, given that you now have automation, you might write additional tests that wouldn't have been viable in a manual environment. Finally, what value do you put on improved time to market and improved quality?

Finally, let me finish with a case study of a project I recently worked on. The software under test was a mixture of IDE (an Integrated Development Environment which included a compiler for an extended C language), code running on a host server (under various favors of Linux and Windows) driving an external peripheral and a number of host applications using the peripheral for support functions.

- The automated test environment was integrated with the code configuration management and build system. We were also able to structure the tests in line with the software so that as the software was built we only tested the changed code. This significantly reduced the test execution time and the test result analysis time and allowed us to perform a continuous overnight build and test cycle. Thus we were able to track the stability of the software, which not only meant improved time to market but enabled better estimates of the time required to stabilise the software after significant changes and enhancements.
- The automation framework was developed in-house by the same team that had developed the in-house software version control and build system.
- We mainly automated the unit level tests and integration testing of the driver and host application code with peripherals across the range of platforms.
- We outsourced the execution of the remaining tests. These tended to be higher level customer focused tests, which we found harder to automate.
- We also outsourced some of the automated tests, but did find it was harder to import the tests into our automation environment as the outsource organisation did not have knowledge of our test automation tool.
- We integrated a static analysis tool into the build system. This found a number of potential issues which we fixed over time and were able to ensure they stayed fixed by regular running of the tool.
- Test maintenance was difficult, and we found that we lost knowledge regarding the older tests as people moved around or left the organisation. This can present significant problems when the test fails. This is not the case with manual (or semi-automated) tests, as somebody is always responsible for maintaining and running the tests. However, ownership and handover of automated tests was not so visible and well managed.

The automation used in the above project significantly improved time to market of the software: we were able to shorten release schedules and hit the release dates. The quality of the software (as measured by defects in the field) was also significantly improved mainly through a combination of the automated testing and outsourcing of customer- focused testing.

This article has demonstrated both the advantages and issues surrounding the automation of software testing. It has attempted to provide practical solutions to those issues and demonstrated how the automated testing can improve time to market thus allowing the test team to demonstrate a significant business advantage

References

1. "Static Analysis Demand to Drive the Market for Test Automation Tools", Venture Development Corporation, 2008 (see http://www.vdcresearch.com/_documents/pressrelease/press-attachment-1413.pdf)
2. "Architectures of Test Automation", Cem Kaner, 2000 (see http://www.kaner.com/pdfs/testarch.pdf)

Biography

Mike graduated from Bristol University with a PhD in Mathematical Logic. Since then he has studied with the Open University obtaining an MSc in Software Engineering and an MBA. He has been involved in both software testing and hardware verification for about 20 years. He started in software testing specialising in formal methods, before moving to verify hardware at ST Microelectronics and Infineon. Most recently, as Test and Verification Manager at Elixent (now Panasonic) and ClearSpeed in Bristol, he has been responsible for the sign off of complex hardware/software products. Mike has had numerous papers published, presented at a number of conferences, and has written on software testing for the Open University. He has been involved in a number of successful outsourcing relationships.

Automating the testing of the GUI for Multilanguage applications

by Marco Torres

Overview

GUI tests are an important part of the Software quality assurance process because they are performed from the point of view of the end users of the application. However, manual GUI testing is time-consuming, labor-intensive and therefore expensive. For testers it is boring, tedious and therefore error-prone.
Here, an automated test suite is recommended as the most efficient method to execute the testing of the GUI elements of multi-language applications.

Introduction

An analysis of bug reports at Microsoft demonstrated that, for localization, almost 80% of the bugs were due to cosmetic issues. This was a good example of the Pareto principle, also known as the 80/20 rule – 80% of the bugs were due to 20% of the causes [1]. An analysis of issues reported by Globalization test teams at Citrix shows similar results.
If we could automate the finding of even half of these 80%, then testing and reworking costs would be significantly reduced.

An automated test suite is an efficient method to execute the testing of the GUI elements. The key points of the proposed methodology are as follows:

- Automated test scripts are language and OS version independent. This allows simultaneous execution of UI test in all languages and supported versions of OS.
- Automated tests will find more cosmetic bugs than manual tests. Issues that are hidden to the eyes of manual testers can be identified by automated tests.
- Improve test coverage. More tests can be executed in the same timeframe. Increased test coverage improves the chance of finding defects that might otherwise make it into production, therefore increasing product quality and not just productivity.

Automated UI Test

The strategy for the automated UI test is simple: To navigate through all windows of an application while executing the following UI tests on each window displayed.

- **Text truncation**: Verify that text contained in the control is completely displayed.
- **Access key (hotkey) clashes**: Verify that each access key is unique within a given window.
- **Overlapping of controls**: Verify that controls are not overlapping or out of boundaries.
- **Alignment of controls**: Verify that controls inside a common container are aligned.

Road to success

Test Automation is a software development process. Test Automation is not testing. Therefore, to be successful on implementing automated tests for multi-language applications, the first step is to follow one of the most important rules used in developing internationalized software applications:

Externalize anything that is language dependent. Actually it should be extended to externalize anything that can change (in current or future releases) like window captions or product names.

The second step is to promote the use of:

Data-driven automation. Data is maintained out of the scripts. If something is changed, the data layer will be updated, not the scripts.[2]

Unique names for controls. A unique name makes it easier for both the script and the tester to identify the controls.

Recognition of controls as objects. Then you have access to its properties and methods, for example:

```
Button.Name(NextButton).
Caption <- Properties
Button.Name(NextButton).
Click <- Methods
```

Properties that uniquely identify each control (like Name in .NET applications).

And the third step is to avoid the use of:
Bitmap comparisons as they are OS version and environment dependent. Instead use object comparison (properties)
Coordinates to interact with controls, for example:

```
Window1.Click(X,Y)
```

Labels/captions to recognize controls as they are language dependent.
Hardcoding entries on scripts (the most important).

To be efficient the Automated UI Test must satisfy the following requirements:

- **Language independent**: The same test script is used in all languages supported.
- **Site independent**: Scripts developed on a given site must run at any other site without modification.
- **OS version independent**: A test script developed in one version of OS (for example Windows XP) must run without modifications in another version of the same OS (for example Windows 2003).
- **Simultaneous execution**: Execute the same test simultaneously on different machines/environments/languages.

Let's analyze how to satisfy these requirements.

Language independent

Test scripts must be written from the start considering possible future language extensions. Initially, develop the script to support the English platform, then create translation tables (external to scripts) that provide the strings in the required language.

Control_Name	English	Spanish		
Disconnect_Button	&Disconnect	&Desconectar		
Print_Button	Print	Imprimir		
Settings_Window	Settings	Configuracion		

Control_Name	English	Spanish	Japanese	
Disconnect_Button	&Disconnect	&Desconectar	切断(&D)	
Print_Button	Print	Imprimir	印刷	
Settings_Window	Settings	Configuracion	設定	

You need to add a language to the test? Just add the relevant strings to the Translation Table. No need to modify the scripts.
The following techniques can also be used:
Name Mapping. Assign a custom name to a control and specify property values that will be used for recognition of this control. After mapping an object, it can be addressed in scripts by its custom name. If developers change the object's properties, name mapping settings may need to be modified, but not the script.
Name property. Use "name" property of controls to interact with them (instead of "caption" or "classname" properties that could change with platform/language).
This will help to keep the size of the translation table to a minimum, therefore reducing the maintenance effort.

Site independent

Externalize any data that is site dependent, like Server names, domain, account name, password, file locations, etc. External data (in the translation layer) can be modified to match the conditions of the test environment.
One common mistake in implementing automated tests is to use fixed delays to allow a task to be finished.

```
OpenNewWindow.Click -> Click on
button "OpenNewWindow"
Delay(1000) -> Wait 1000ms until
NewWindow is displayed
Next Action -> Proceed with next
action
```

Although this approach can work for the development environment, it is likely to fail in the test environment (if **NewWindow** is not displayed in less than 1000ms).
Instead of fixed delays use "WaitFor" instructions that allow the script to wait until some specific condition is satisfied, for example:

```
NextButton.Click
if WaitForWindow(Window_Name,
1000) then
      Next Action
else
      Log.Error(Window_Name+"
was not found")
```

WaitForWindow waits until 1000 ms for Window_Name (returning as soon as Window_Name is displayed, eliminating in this way the waste of time produced by fixed delays). If the Window is not displayed in the time expected, the script fails gracefully reporting a "Window_Name was not found" error.

OS version independent

Data that is dependent on machine environment will be kept externally from the test scripts, for example images, location of program files, etc.
Use System variables for PATH, etc.

User interface change

Keeping up with design changes in the user interface is a major complication of GUI test automation [3]. Telling programmers they should freeze the user interface doesn't work. The following techniques can help to reduce the impact of changes in the GUI:

- **Abstract the interface**: If the user interface changes, the abstraction layer will be updated rather than every test that accessed the changed interface.
- Use an **object-oriented recognition model**: If, for example, the position of a control changes, there is no need to modify the script. However, the script may fail if the control's recognition attributes are changed (for instance, the button's caption is changed).
- Use of **metacharacters** for recognition, for example:
- Product_Release_1 -> Product_Release*
- Use of Name Mapping (already mentioned in Language Independent section) can also help to reduce the impact of changes in the UI. If developers change the object's properties, name mapping settings may need to be modified, but not the script.
- **Fallback to English**. If a control search using a localized string fails, the script will use the English string assigned to such control in trying to recognize the required control.

```
LangID=GetLangID
winName=Settings_Window(LangID)
w1=WaitForWindow(winName, 100)
if w1.Exists=False then
      Log.Error("Localized Win-
dow not found")
      winName=Settings_
Window(EnglishLang)
      w1=WaitForWindow(winName,
100)
end if
```

LangID get the identifier of the localized language, then try to get an object for "Settings_Window" in that language; if it fails, an error message is produced to indicate that localized Window was not found. In a second attempt to get the object, the value in English language of "Settings_Window" is used.

Testing the UI

The starting point of testing an application GUI's is to locate the GUI components. A Test Automation tool that recognizes these components programmatically provides a reliable approach and is not affected by changes in control position, order or addition of new controls. [4]

In general, all UI tests can be implemented in three steps:
1. Identify the controls in the window.
2. Retrieve some properties.
3. Verify that values of properties satisfy a given criteria.

UI Test –Alignment-

The purpose of this test is to verify that all controls of the same type inside a common container are aligned. For example, we could verify the alignment of Radio buttons in the following way:

```
Left_Position=RadioGroup1.
Controls(0).Left
for i=0 to RadioGroup1.Controls.
Count-1 do
      if RadioGroup1.
Controls(i).Left <> Left_Posi-
tion then
      Log.Message("RadioGroup1.
Controls(i) is misaligned")
```

First, set the value of Left_Position as the reference, then verify that all controls in the group match the reference; if any control doesn't match, then report the error.

UI Test –Access keys-

This test verifies that each Access key is unique within a given window.
The basic idea is to create a list of access keys by "enumerating" all controls in the window and selecting those that contain "&" (ampersand symbol) in Caption/Text (access key is the character after the "&").Try to add each access key found to the access key list; if the access key is already in the list, then we have found a duplicate access key.

UI Test –Text truncation-

This test verifies that text contained in the control is completely displayed.
Let's first define what a text truncation is. Text is displayed inside a container; both the text and the container have width and height.

Basically, if the width of the text is larger than the width of the container we have a truncation; additionally if the height of the text is greater than the height of the container, there is also a truncation, e.g.

```
if Text_Width > Container_Width
OR
```

```
Text_Height > Container_Height then
              Text_Truncation=True
```

There are several methods to verify the text truncation.

- For Windows applications, we can use the API function DrawTextEx that draws formatted text in the specified rectangle (the container); by using the option DT_PATH_ELLIPSIS (for displayed text, replaces characters in the middle of the string with ellipses so that the result fits in the specified rectangle), we can get a modified string if the text doesn't fit in the container. Therefore, if the string "before" applying DrawTextEx is different from the string "after", we have a truncation.


```
Text_Before <> Text_After -> Text Truncation
```

- Another method is to use Autosize property of controls (if available) In this case we can compare the size "before" applying Autosize and "after". If the size of the control after applying Autosize is larger than "before", we have a truncation.


```
Width_Before < Width_After -> Text Truncation
```

Results

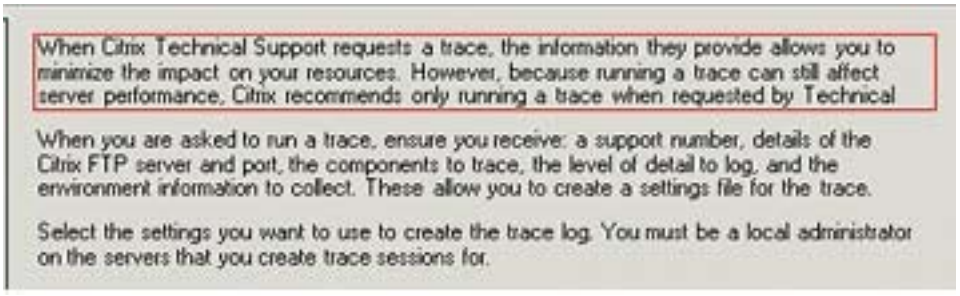
Execution of the Automated UI Test Suite for testing an application that supports 5 languages (English, German, French, Spanish and Japanese) produced the following results:

- Around 800 windows were tested
- More than 50 cosmetic bugs were found, which were not detected by manual testers
- The execution time was 12 hours
- Adding a new language to the UI test suite took 2 hours

Below are some examples of bugs reported by the Automated UI Test that couldn't be detected by manual users.

Duplicated Hotkey (Alt-d)

Missing Text



Actual Text: When Citrix Technical Support requests a trace, the information they provide allows you to minimize the impact on your resources. However, because running a trace can still affect server performance, Citrix recommends only running a trace when requested by Technical **Support.**

RadioButtons misaligned (see screenshot below)

Conclusions

Although the automated UI test will be initially more expensive than the equivalent manual testing, automated tests can be executed very cheaply and will thus result in cost savings with repeated runs over time.

As shown in the Results section, automated tests were able to find many issues that manual testers couldn't see.

Execution cost is reduced since only one person is required to execute and maintain the automated UI Test suite for all languages. Manual UI tests require one person per language.

The proposed methodology has been successfully implemented on Windows, .NET and Java applications and tested in the following languages: English, German, French, Spanish, Japanese, Chinese, Korean and Russian.

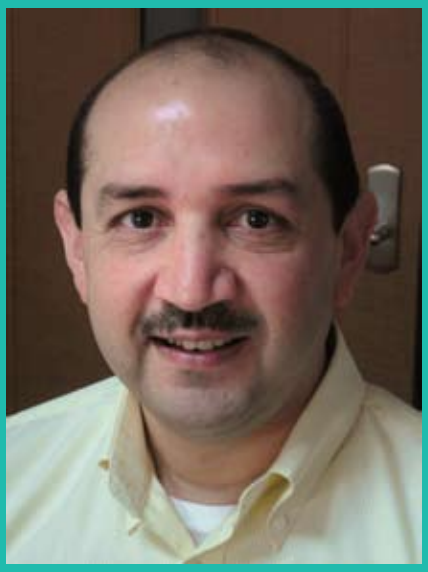
References

[1] Software Test Automation. Fewster & Graham pp. 544

[2] Totally data-driven automated testing. Keith Zambelich

[3] Lessons learned in software testing. C. Karner, J. Bach, B. Pettichord pp. 101

[4] Effective GUI Test automation. Kanglin Li, Mengqi Wu pp.10



Biography

With a Ph.D. from Nagoya Institute of Technology in Computer Science, Marco has more than twenty years of experience in leading all aspects of successful software development, including support, development and testing, specializing in the areas of software testing tools, techniques, and test automation.

In his current position at Citrix Systems Japan R&D, he is responsible for developing and implementing a long-term test automation strategy for the next generation software products.
He is focusing on creating low-cost test automation architectures that are easy to implement, easy to use and easy to maintain.

Thank You!

TESTING DAYS GERMANY 2008

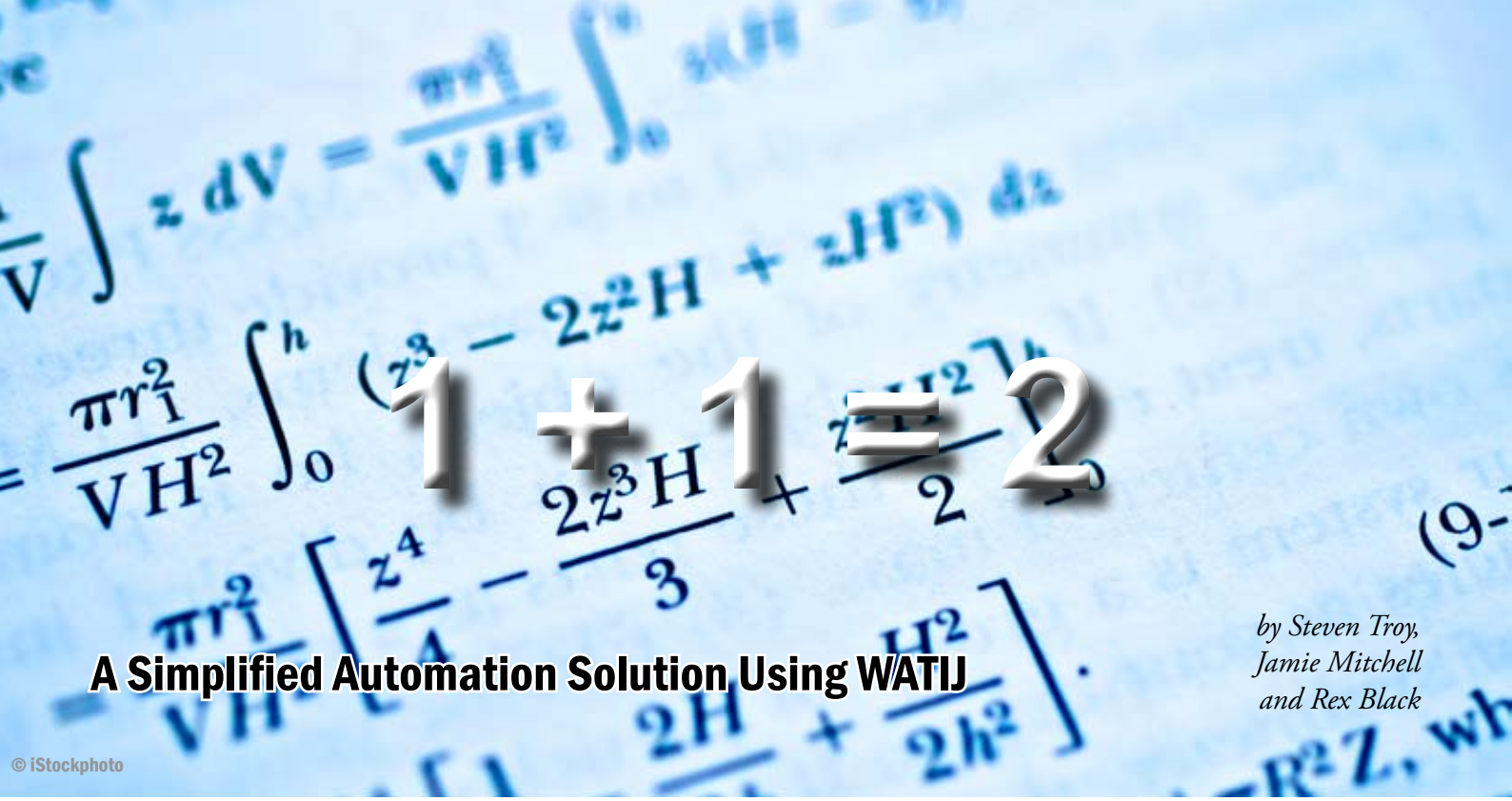
Many thanks for participating!

We are very pleased that so many testing professionals found the time to participate.

Consequently, we are looking forward to welcoming you again next year at our road show "Testing Days 2009"

www.compuware.de

Compuware



A Simplified Automation Solution Using WATIJ

by Steven Troy, Jamie Mitchell and Rex Black

Regression is a serious risk for software. Unlike physical structures such as bridges, skyscrapers, and submarines, software is quite sensitive to small changes, which can result in serious, unpredictable side-effects, often functionally distant from the change that was made. Because of this, test teams use regression testing to mitigate the risk.

Unfortunately, the scope of regression testing grows in a non-linear fashion over the life of an application. While the scope of a maintenance programming activity—say, to release a few bug fixes and features for a successful existing application—is linear in relation to the number of bug fixes and features, the scope of the regression testing includes all the bug fixes and features ever done.

This leads many organizations to try to automate regression testing. Sadly, many organizations fail in these attempts, either because of high initial costs or because of maintainability problems with their regression tests. Lack of good design for the automated test frameworks and bad expectations on the part of people involved often contribute to these failures.

Understanding Our Regression Testing Problem

In this article, we'll talk about how we implemented automated regression testing in a maintainable, low-cost way for a widely-used product called CA Service Desk Manager. This popular, versatile, and comprehensive IT support system is browser-based. Therefore, we crafted an extensible automation architecture for CA Service Desk Manager that we can use in a wide variety of our future quality assurance needs. We are still developing this test system, but we already have over 2000 test cases running dependably and we are adding more daily. We call our solution the Service Desk Manager Automation Framework Code.

CA Service Desk Manager is a business critical, high-availability product. Customers and users insist on the utmost reliability. Downtime costs money. Our QA group, responsible for regression testing this system, previously had relied on over 13,000 manual test cases to ensure high quality during maintenance testing. We needed a new approach to test this application, since each maintenance release and patch required a tremendous manual test effort. As mentioned above, we knew the amount of regression testing effort would only increase.

While it was clear that test automation would need to be part of the solution, we knew that automation raised a number of problems on its own. We'd had a lot of past experience with automation, mostly unsatisfactory. The supposedly simple tools were not simple! The record/playback tools that supposedly allowed testers without programming skills to easily create, run and maintain libraries of automated test cases didn't deliver on that promise.

Our challenge was to come up with a solution which we could support, extend and utilize without an exorbitant amount of effort. In addition, many of our testers do not have formal programming experience; they are domain experts who came from the business side of the organization. Replacing established testers with programming-skilled tester who could directly script tests was not an option, nor was trying to morph all of our testers into programmers.

We understood that we would still need automators - developers who were experienced with programming languages and associated tool sets. We decided to utilize our automators to create a meta-language, to use an advanced tool set to build a simpler tool set that our testers could use directly. Done right, a few automators should be able to support an unlimited number of testers. Since the testers would build the needed assets, the test cases, we saw this as a way to give our solution scalability. The growth of our test cases would not be limited by the number of automators but by the number of testers; this mirrors manual testing. Essentially, we wanted to use our automators to add complexity to the heart of our test system to make the interface of it simpler. In other words, we would design our test system for usability by the manual testers, following the Keep It Simple, Stupid (KISS) principle.

Selecting the Right Tool

This principle led us to the following mandates for our test framework:

- Make it relatively easy to develop, since we don't want to build a more complex test framework than the system under test (SUT).
- Make it easy to maintain.
- Make it easy to use by any tester. (We joked that it needed to be easy enough for a manager to use.)
- Make it powerful and efficient enough to automate a large number of test cases within a reasonable time.

For the heart of our system we needed a toolset for the automators to use. We considered both commercial record/playback tools with framework tools that sit on top of them and open source scripting and automation test tools. We quickly came to prefer a tool called Web Application Testing In Java (WATIJ.) This tool is an application programming interface (API) wrapped around the Java language, expressly created for the automated testing of web applications through Internet Explorer (IE.)

Since this tool would put Java at our automators' fingertips, we would have access to an enormous number of existing libraries and tools that our automators could use. A large benefit of this synergy would be that our test system would not be restricted to only testing

web applications; we could use it in the future to expand to related areas that need testing. For example, we would be able to write a test that accesses a web service directly, calling on it to do some processing. We could then use WATIJ to bring up CA Service Desk Manager to validate that the processing behaved as expected. In addition, since parts of our system under test are written in Java, WATIJ would fit in nicely to our existing environment.

We have to perform much of our testing at the graphical user interface (GUI) level, so our framework needed to interact directly with the CA Service Desk Manager interface. Since WATIJ only supports IE, this limited our automated testing. We decided, however, since most of our testing was already done in IE, we could live with this constraint. There is reason to believe that, in the future, WATIJ will also directly interact with Firefox and other popular browsers. Until then, we could automate the majority of our tests using IE and then use a subset of our manual tests to validate CA Service Desk Manager on other browsers.

Another limitation that initially concerned us was the lack of record/playback capability in WATIJ. Upon reflection, we realized that as long as our automators could supply abstract interfaces (essentially keywords) to our testers, this constraint would not reduce the benefits we would obtain.

Using WATIJ, we would be able to simulate human user interaction with the SUT. This includes simple tasks that any automation tool could do, including page navigation, clicking on links, filling out forms and validating form content. An advantage of WATIJ is that it also facilitates much more complex actions, such as file uploading and downloading, handling popup windows, dialog manipulation, and screen captures.

Our automation framework needed to be able to find disparate HTML elements on a page. Many of the tools that we investigated did not handle this sometimes cumbersome task well. We were delighted to find that WATIJ included a powerful element finding capability that can find, access and control any HTML element easily. It even supports XPath expressions that are lightning-fast.

A final concern was the programming environment for our automators. Because it is configured as a standalone API, WATIJ programmers are not forced into using a proprietary testing framework or special integrated development environment (IDE.) Instead, we could integrate WATIJ into almost any popular testing framework we wanted, including JUnit or TestNG. As WATIJ works like any other Java library, it could also be imported into various popular IDEs, including Eclipse and IntelliJ. This easy integration should ensure high productivity immediately without special training or ramp-up time for our automators.

We finally decided to use WATIJ rather than other open-source or vendor tools based on the following considerations:

- WATIJ is open-source with a robust community of users.
- WATIJ is extremely fast to write as well as execute.
- WATIJ provides automatic synchronization, since WATIJ waits until a page download is complete rather than trying to manipulate partially downloaded pages.
- WATIJ is easy to use.
- WATIJ is Java-based, giving us the integration benefits noted above.

At this point, we believed WATIJ would provide the right capabilities for our test framework.

Implementing the Framework

Having picked our tool, we understood that we were only half-way there. We had a solution for our automators, but nothing yet for our testers to use. Our next step was to determine which low-level GUI interactions were the most common. By understanding the GUI domain, we could prioritize the logical automation capabilities we needed to build into our framework.

We split our GUI domain into two separate sets of tasks. First, we isolated general low-level control interaction tasks that testers would need in any browser application. These included:

- Click on a link
- Click on a button
- Set a text field
- Set/unset a radio button
- Check/uncheck a check box
- Select a value from a drop-down combo or list

We decided that these tasks could essentially be distilled down into two abstract functions:

- setItem()
- clickItem()

When called, these two functions would "do the right thing" based on the control passed to it. Remember, we were creating a simplified language (called the Service Desk Manager Framework Code) that non-technical testers would use, directly creating automated tests. Wherever possible, we wanted to avoid any low-level commands in this language, but we knew we would need them sometimes.

A common task that needs to be performed is logging into a web site. As an example, consider the Yahoo login shown in Figure 1.

Figure 1: Web site login

The manual tasks that a user needs to perform to login are:

1. Bring up a browser window
2. Set the correct URL and open that page
3. Wait for the page to load and stabilize
4. Type the user name into the ID field
5. Type in the password to the password field
6. Click on the Sign In button

The following programmer-written WATIJ code directly performs these tasks:

```
IE ie = new IE();
ie.start("www.mail.yahoo.com");
ie.textField(SymbolFactory.id,
"username").set("Harry_Potter");
ie.textField(SymbolFactory.id,
"passwd").set("Alohomora");
ie.button(SymbolFactory.id,
".save").click();
```

Here are the simplified commands that the tester would write:

```
start("www.mail.yahoo.com");
setItem("Textfield", "username",
"Harry_Potter");
setItem("TextField", "passwd",
"Alohomora");
clickItem("Button", ".save");
```

Because the system automatically handles synchronization, the tester need not worry about it.

However, the above example does not really show the simplification that we get from this framework. We get much more leverage when dealing with more complex tasks that are specific to the SUT, CA Service Desk Manager. This is the second set of tasks that we needed to perform. Each task is encapsulated in a keyword-like method written as part of the Service Desk Manager Framework code. Examples include the following:

- clickNode() // Click on Any Node in the User Interface
- login() // Login to Service Desk Manager
- logout() // Log off of Service Desk Manager
- alertMessage // retrieve a message from alertmessage
- isLookup // checks whether a link is a lookup or not

Let's look at an example of how this works. Figure 2 shows the screen for logging into the CA Service Desk Manager.

Figure 2: Logging into the CA Service Desk Manager

Consider performing the following tasks against this interface:

- Open a browser window
- Bring up and initialize the CA Service Desk Manager application
- Log into it
- Move to the Administration tab
- Select a node in the tree

Here is the WATIJ code as created by a programmer:

```
IE ie = new IE();
ie.start("http://servicedesk:8080");
ie.textField(SymbolFactory.name,"USERNAME").set("Harry_Potter");
ie.textField (SymbolFactory.name,"PIN").set("Alohamora");
ie.button(SymbolFactory.name,"imgBtn0_button").click();
ie.frame(SymbolFactory.name,"toolbar").link(SymbolFactory.id,
"tabhref2").click();
ie.frame(SymbolFactory.name,"product").frame(SymbolFactory.
id, "tab_1004").frame(SymbolFactory.id,"role_main").
frame(SymbolFactory.id,"MenuTree").frame(SymbolFactory.id, "frmAd-
mTree").div(SymbolFactory.id,"scrollbarDiv0"). div(SymbolFactory.
id,"divTreeNode1"). link(SymbolFactory.title, "Security and Role
Management").click();
ie.frame(SymbolFactory.name,"product").frame(SymbolFactory.
id, "tab_1004").frame(SymbolFactory.id,"role_main").
frame(SymbolFactory.id,"MenuTree").frame(SymbolFactory.id, "frmAd-
mTree").div(SymbolFactory.id,"scrollbarDiv0"). div(SymbolFactory.
id,"divTreeNode1"). link(SymbolFactory.title, "Contacts").click();
```

And, here is a simple series of commands, using the Service Desk Manager Framework, that a tester would write:

```
Start("http://
servicedesk:8080");
Login("Harry_
Potter","Alohamora");
clickItem("toolbar","Link","tabh
ref2"); //Administration Tab
clickNode("Security and Role
Management->Contacts");
```

Once again, the framework handles the identification of GUI elements and synchronization, freeing the testers from having to deal with that complexity. They simply write short declarations of abstract tasks performed in the same order that they would have performed manually.

A tester can also use the text of the link or button to identify the GUI element as well. In the above example, to open the Administration tab, the tester could have written:

```
clickItem("toolbar","Link","Adm
inistration");   //Administra-
tion Tab
```

Some interactions with the interface are not necessarily tied in with specific GUI elements. In these cases, our framework allows the tester to submit keystrokes the same way a manual tester might do manually. For example, to open the file menu of the browser, a manual tester might simply key in Alt-F. In our framework, we can perform exactly the same action by typing in the following statement:

```
sendKey("%f")
```

Our framework now had everything we needed for a tester to create a test case that initialized the SUT and cause it to perform a series of actions, driving through the test case. What we were still missing, however, was a way to make it a real test. Each test, to be valid, must have a way to compare actual behavior with expected behavior.

To that end, our automators have added to the framework numerous validation methods. For example, suppose we want to check to ensure that a field is set to a specified value after the test is run. The actual WATIJ code we use is:

```
public static String getCell-
Value_id(String _framename,
int _table, String _id) throws
Exception
{
String ret = null;

try{
if(((HtmlElement) ((HtmlElement)
findFrame(ie,_framename).table(_
table)).cell(SymbolFactory.id,_
id)).exists())
{
ret=(String) ((HtmlElement)
findFrame(ie,_framename).table(_
table)).cell(SymbolFactory.id,_
id).text();
}
else
{
_teststatus = "fail";
}
catch(Exception e)
return ret;
}
```

This code is then encapsulated in the Service Desk Manager Framework as follows:

```
Get_value=getCellValue_id("cai_
main", "dtltbl0",df_0_2").trim()
.equals("3"));
```

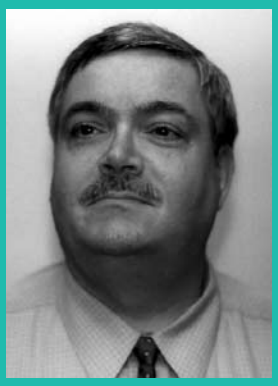
As you can see, this simple command provides our testers with access to a powerful capability without having to master the programming complexity underneath that capability.

Conclusion

While we are still implementing our framework, we can already say that our overall approach to automation using WATIJ as the backbone of our framework has been a great success. We have already automated over 2,000 test cases, fully 15% of the total we will need and a large enough sample to prove the soundness of our approach. We are very pleased with the work of Margie Studer, Chris Whorley, Hari Maddela and Sujit Karri, the architects who made the Service Desk Manager Framework a reality. We are also very happy with the accomplishments made by our entire QA team, who are successfully using framework.

Biography

Steven Troy is a Senior Director of Quality Assurance for CA, Inc. (www.ca.com), the world's leading independent IT management software company.



Jamie Mitchell of Rex Black Consulting Services is a senior consultant specializing in testing, training and automation.

Rex Black is President of RBCS (www.rbcs-us.com), a worldwide leader in testing services, including consulting, outsourcing, assessment, and training.

Knowledge Transfer

testing experience

Limited places

Tutorial

Agile Inspection Leader & Value Requirements

with certification

by Tom Gilb & Kai Gilb

2009, March 16-18 in Munich

Agile Inspection Leader (1 day)
- How to perform an efficient Agile Inspection.

Including: Using Inspection to measure document quality, Sampling, How to write and inspect Large Documents, Inspection on Requirements & Designs & Code & Tests, How to use data to optimize your inspection, Optimum Checking Rates, Defect Types. Defect Density, Numeric Entry & Exit Criteria.

Qualifies participants for certification: Agile Inspection Leader - Trained
For more information see: http://www.gilb.com/Inspection+Leader+Certification

Value Requirements (2 Days)
- How to Specify Critical Stakeholder & Product Value Requirements.

So they are: Agile, Clear, Meaningful, Quantified, Measurable and Testable. Specified so they are representative of the real requirements that your many Stakeholders have; and specified in such a way that managers/engineers/developers can intelligently prioritize them and develop towards satisfying them.
Stakeholder Value & Product Value Requirements are of the type that is both critical for success of any project, and of the type that most engineers do not know how to specify clearly and quantitatively. A certified Value Requirement professional adds benefit to any project by knowing how to identify, specify and focus a project towards satisfaction of Critical Stakeholder Value & Product Value Requirements.

- How to Specify core Function Requirements.
Many developers, especially in software development, treat Functions, or what they normally call Functionalities - sometimes specified as use cases or stories, as the dominant type of requirement. This is clearly not representing reality well, yes functions are fundamental, in that they specify what your product or service must do to be on the market at all. But Functions are pretty much the same among all your competitors. Functions do not, by themselves, make a product or service competitive, they are only interesting in combination with Product Values (qualities).
Value Requirement professionals get insights into what Function Requirements really are, how to specify them, and how to use them effectively in combination with Stakeholder and Product Values.

- and they learn, How to Specify Solution Requirements, and how not to mix solutions up with functions.

Qualifies participants to certification: Value Requirements Professional - Trained
For more information see: http://www.gilb.com/Value+Requirements+Certification

1 day - Agile Inspection Leader	850,- EUR*
2 days - Value Requirements	1350,- EUR*
3 days - both courses	1850,- EUR*

Register till 2009, January 30th you get 15% discount!

Please register by email kt@testingexperience.com or fax +49 30 74 76 28 99

* plus VAT

Ensuring SOA ROI

by Rami Jaamour

Introduction

As more and more Web services integrate into core processing systems, the reliability and performance of these Web services becomes a primary goal for businesses. Companies must ensure that the Web services deployed are as sound as possible, because they serve as the backend driver for business processes such as customer-facing applications, CRMs, SCMs, etc. Outages for such critical components could cause companies large losses due to business disruptions.

Therefore testing, which is important for any Web application, is even more crucial for Web services and Service-Oriented Architectures. Extensive testing of Web services, particularly those that are externally facing and business-critical, is essential for protecting the enterprise from significant business risks. A downtime of an hour might not only cost substantial losses in revenue but, more importantly, the perceived lack of quality and reliability of the company in general. Any mission-critical Web service must be strictly tested and verified as functioning correctly, 24 hours a day, seven days a week – no exceptions.

This paper will explain issues specific to Web services and will illustrate the best practices that can ensure a secure, reliable, and compliant SOA.

Web Service Testing – Why Focus on Testing?

Web services design and development tools have evolved to a point where there is little code needed in order to put the basic Web services infrastructure in place. IDEs like eclipse WTP and STP have the ability to visually design WSDLs, Schemes and then generate the required code templates at the click of a mouse. Furthermore, the evolution of ESBs and orchestration engines take this further by automatically exposing a variety of legacy functions into XML-based Web services, all in a manner where SOA Architects can point and click to configure the processes and transformations they need.

Once the basic skeletons have been generated and put in place, evolving these skeletons with the necessary logic is often a mix between traditional software development efforts and visual process design within SOA development tools. However, there is relatively little said on what needs to happen in the service SDLC in terms of a quality process which addresses the unique challenges surrounding Web services and which ensures that they meet their end goals for security, reliability, compliance and interoperability.

Common Web Service Testing Problems

There are many complexities specific to, and inherent in Web services that complicate testing. However, before delving into specific technical issues, it would be wise to first examine the general testing inefficiencies that can occur in any Web service. There are a number of testing inefficiencies that plague companies and have severe implications on business such as extending project timelines and increasing project costs.

Inefficient Use of Developer Assets

The extent to which organizations continuously re-create the same test cases at different points of the development process is surprising. Developers create tests to ensure the basic functionality of their Web services; however these tests are not leveraged in the downstream process. This duplication of effort is not only inefficient, but also introduces an opportunity for errors to be injected.

Development and Maintenance of Homegrown Tools

Tools, such as simple Web-based forms for submitting XML requests, are developed in-house in order to facilitate functional testing. At the same time, a load generation tool is purchased that requires proprietary scripts to be written, without leveraging the work done in functional testing. The creation and use of such homegrown tools and scripts not only adds overhead to the project, but such tools are also extremely costly to maintain as industry standards and Web service complexity evolve and mature. Besides, they do not provide an infrastructure for a repeatable, automated regression testing process that facilitates reuse.

Incomplete Testing

The scope of testing is generally limited and does not cover all the aspects needed to verify a Web service. Aspects such as security, interoperability, functionality, reliability, and availability are not verified due to these limitations.

For example, rather than testing a Web service directly at the messaging level, organizations may rely on testing merely through the application which consumes the service, perhaps via its Web-based interface. This approach restricts the service tests only to those types of messages which the GUI-based application is built to generate. As Web services are reused in a SOA environment, it is unfeasible to anticipate exactly what types of requests the various clients will send. Furthermore, the approach does not address rogue messages that may be crafted by a malicious user. Real-world partners and consumers of a Web service are certain to send different types of requests, negative inputs, and bad data to the Web service that the GUI-based client cannot produce because it is constrained by scenarios and the interface that has been designed around them. In order

to completely and thoroughly test a Web service, you must be able to emulate many of the types of client that might access the server, and verify that the server will behave as expected in relation to any type of client request.

Best Practices for Verifying and Testing Web Services

In order for a complete Web service to deliver optimum functionality, both the client and the service must satisfy a number of requirements. Interfaces must be correctly described in a WSDL document. Messages must conform to both the transport protocol specification (such as HTTP 1.1) and the message protocol (such as SOAP 1.1). Mes¬sages must also conform to the contract specified in the WSDL describing the service, both in terms of the message content and the binding to the transport layer. Add to the mix security provisions, interoperability issues, and performance requirements under load, and it is easy to see why Web service testing is not a trivial matter.

WSDL Validation

WSDL validation can be considered as the first step in testing Web Services. Although WSDLs are generally created automatically by various tools, it doesn't necessarily mean that they are correct. When WSDLs are manually altered, WSDL verification becomes even more important. As a minimum, WSDLs should be checked for conformance to the WSDL scheme via XML validation. Additional checks include interoperability conformance checks against WS-I's Basic Profile and semantic validation.

Unit Testing at the Service Operation Level

Following the validation of the WSDL, the next step in testing a Web service is to create unit tests. Each module of a Web service should be developed along with unit tests that verify its functionality. By requiring the development of unit tests before or during, and not after, the development of the code, it can be ensured that the code actually provides the necessary functionality. The scope of the unit tests should be such that every functional requirement of the module is verified. In addition, by using unit tests to drive development, unnecessary code that does not need to be written can be avoided. However, unit testing can be quite labor-intensive if performed manually. The key to conducting effective unit testing is automation.

Regression Testing

Regression testing ensures that what worked yesterday continues to work tomorrow, and that functionality is not broken as the services are evolved in time. Once a unit test meets a requirement, it is frozen, and we can move on to the next requirement. The way to make sure that a requirement stays frozen is to check the unit tests into source control or test management system and run them automatically as regression tests during the nightly test process. This means that once a unit test passes, it should always pass. There will be times when a change to a seemingly unrelated part of the code causes undesirable behavior in another part of the code. By having these unit tests that serve as regression tests, undesirable changes over time can be detected. As the functionality becomes increasingly complex, previous tests should continue to pass and new tests that test new functionality should be added.

Functional Testing

Unit tests created by the developers can be leveraged and extended into more complicated functional and scenario tests. This is accomplished by combining a series of operations into a complicated scenario that emulates potential client behavior.

Performance Testing

Once functional tests have been developed, the Load Generation Team can specify the performance objectives for all the test cases produced by QA and Development and generate the load for the servers, rather than having to spend months writing additional scripts to facilitate their testing. Load testing not only monitors the server's response rate with the specified number and mixture of simultaneous requests, but also verifies whether the test loads cause functionality problems.

Stubbing and Service Emulation

Developers of service-consuming applications are responsible for making sure that such applications send correct requests and respond to the service messages as expected. As services evolve in a composite applications environment, emulating different components of the system becomes critical for making these systems testable. The reason is that SOA is distributed by nature, and services are often maintained by multiple distributed teams. This makes the process of setting up a test environment complex. Therefore, there is a need to create reusable, sharable service emulations that can fill in the gaps in a system and allow for the business processes to be executed as part of end-to-end testing.

Web Service Interoperability

Interoperability is key for achieving service reuse in a SOA. Performing functional testing, load testing, and client testing will ensure that your Web service functions properly without error. However, as Web service development continues to mature, the issue of interoperability must also be addressed during the testing process. After all, interoperability is really the main reason to use Web services in the first place, since Web services allow for the integration of disparate entities.

Web Service Security

Web services provide a window deep into business applications. They expose a wealth of data and functionality that touch the core of the business. Web services can expose customer, financial, and operational systems to security vulnerabilities through common software flaws such as XPath and SQL injections and XML bombs, many of which can be exercised by insiders even in the case of services that are internal only.

Conclusion

As seen throughout this paper, there are countless opportunities for things to go wrong during Web services development—a slight mistake in any component or interface will cause problems that ripple throughout the system. Developers must ensure that each part of the system is reliable, and that all of these parts interact flawlessly and securely.

Biography

Rami Jaamour is a product manager for SOA Solutions at Parasoft Corporation. Rami has 5 years experience in the SOA and Web services domain. He has published numerous articles and spoken at many SOA-related events, such as RSA, EclipseWorld, SOA World, etc.. His experience with SOA includes the development of effective SOA test automation methodologies and the establishment of effective SDLC processes and activities that are essential for successful SOA initiatives. Rami has worked with many Parasoft clients in achieving their goals of having secure, reliable and compliant SOA.

Root Cause Analysis – Dealing with problems not just symptoms

by Alon Linetzki

Abstract

As test managers, we seek ways to improve in areas such as processes, tools, resources, ROI and so on. In this article I will discuss how we can find the underlying problems within the engineering group (some related to testing, some affecting the testing group in a big way), rather than focusing on the symptoms.

Root-cause analysis and cause-effect graphing (or cause mapping) are known techniques that are used in various ways. I have enhanced them to identify priorities that assist in selecting the best roots for improvement with regard to the implementation and its impact.

Introduction

As a test manager, I have used and exercised RCA (root cause analysis) and CEG (cause effect graphing) a few times in my projects. It always bothered me that I invested so much time in interviews, analysis, questioning, and that I do not get the priorities of the paths and roots which were most probably going to succeed.

One of the fundamental techniques is the "5Ys" or "5Whys". In this technique, we ask ourselves "why" a phenomenon happens, get an answer and then ask "why" again – 5 times – to get to the bottom of things.

The following example demonstrates the basic process:

- My car will not start. (the problem)
-

1. Why? - The battery is dead. (1st why)
2. Why? - The alternator is not functioning. (2nd why)
3. Why? - The alternator belt has broken. (3rd why)
4. Why? - The alternator belt was well beyond its useful service life and has never been replaced. (4th why)
5. Why? - I have not been maintaining my car according to the recommended service schedule. (5th why, root cause)

The questioning for this example could be taken further to a sixth, seventh, or even greater level.

The technique was originally developed by **Sakichi Toyoda** and was later used within Toyota Motor Corporation during the evolution of their manufacturing methodologies. The tool has seen widespread use beyond Toyota, and is now also used within Six Sigma.

There are a few disconnections that can be found in this technique:

- What if one of the 'Whys' is answered wrongly? Maybe our answer is possible, but what if the actual cause is something else?
- Who says the result is the only one derived from the answer of the "why" - There is the assumption that a single cause, at each level of "why", is sufficient to explain the effect in question.
- When we have found the problem, and draw the route, how 'strong' is this solution (how strong is the connection between the source and target phenomenon?) Maybe we should prefer one over the other?

On top of that, at the end of the process I get a nice diagram with connections, (cause-effect diagram/graph), but with no clear observation or a suggestion related to the best root possible, the probability of that root succeeding or the added value of that root by removing the root cause.

What I have witnessed happening today...

In my long years of experience with customers, I have seen that we are doing many things wrong in trying to get it right.

Companies tend to jump too quickly to solutions, not spending enough time on problem analysis.

Usually the quick solutions come up first (sometimes not so easily), but they may not be the best ones (long-term problem eliminators), and often they represent a solution to a symptom(s), but not to the real problem – which we have not identified yet!

I have collected some behaviors which I have observed from the market during the years:

- We pay attention to details, but tend to miss the whole picture…
- We sometimes are too focused on the now and the near future, and do not see enough in the long term,
- We just hate to keep looking for the problem, if we believe we have just found it (or so we believe…),
- We often think we have enough evidence to support the problem identification we have at hand, but we don't…
- We tend to complicate things – otherwise it would not be a challenge to find the problem…

Root Cause Analysis Method – My interpretation

From Wikipedia, I have found a pragmatic definition:

- **Root cause analysis (RCA)** is a class of problem solving methods aimed at identifying the root causes of problems or events. The practice of RCA is predicated on the belief that problems are best solved by attempting to correct or eliminate root causes, as opposed to merely addressing the immediately obvious symptoms.
- By directing corrective measures at root causes, it is hoped that the likelihood of problem recurrence will be minimized.
- However, it is recognized that complete prevention of recurrence by a single intervention is not always possible. Thus,

RCA is often considered to be an iterative process, and is frequently viewed as a tool of continuous improvement.

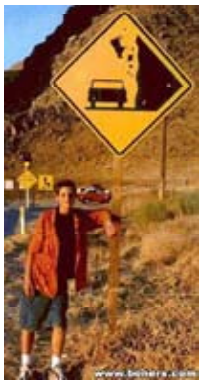
My interpretation and simple definition is:

- RCA is a problem-solving method designed to search for the root of a problem, using drawing symbols and a predefined structural process of thinking.

From Bill Wilson's website:

- *Root cause analysis* (RCA) is a methodology for **finding and correcting the most important reasons** for performance problems.
- It differs from troubleshooting and problem-solving in that these disciplines typically seek solutions to specific difficulties, whereas **RCA is directed at underlying issues**.

Now, to demonstrate how this process of identifying problems is not easy, look at the following picture:



It is a "Cows and rocks are falling from the mountain" sign.

- Is it a symptom or a problem?
- Should we eliminate all cows in that area?
- Should we dig-out the mountain?
- Should we divert the road elsewhere?
- [or maybe] the sign is not posted right…

Sometimes (in my eyes, in most times) eliminating the causes is not an easy task and finding problems merely from symptoms is a hard cognitive process to go through.

Root Cause Analysis Techniques

There are a few techniques being used today. I will mention a few, and will focus only on two I have examined.

1. **Cause and effect analysis** Simplest technique out of all the RCA techniques. For every **effect** there is a **cause**. There is a fairly long chain of relationships between the **cause** and its **effect**. As we move along the chain the cause and effect become finer and finer. Just like when we dig out a tree its roots become finer and finer. The finest **cause** if removed, the problem will not re-appear. This is the essence or Root Cause Analysis.
2. **5 Whys**
3. **Kepner-Tregoe Problem Analysis** - original root cause analysis process developed in 1958, which provides a fact-based approach to systematically rule out possible causes and identify the true cause
4. **Failure mode and effects analysis** Also known as FMEA.
5. **Pareto analysis**
6. **Ishikawa diagram**, also known as the '*fishbone*' diagram or cause and effect diagram
7. **Cause Mapping** - A problem solving method that draws out, visually, the multiple chains of interconnecting causes that lead to an incident. The method, which breaks problems down into specific cause-and-effect relationships, can be applied to a variety of problems and situations
8. **Change analysis** - an investigation technique often used for problems or accidents. It is based on comparing a situation that does not exhibit the problem to one that does in order to identify the changes or differences that might explain why the problem occurred.

Although common methods are: Cause-effect analysis, 5Whys, FMEA, Cause Mapping and fish-bone – I have focused on 2:

- 5Whys
- Cause Mapping

These are commonly used and exercised in the market, and highly understood from my experience.

Enhancing the Method

I have enhanced the method, and present this below using a project implementation.

This is the process I have constructed and used:

1. Conduct structured interviews with customer's personnel
2. Draw the cause-effect diagram and apply the 5Whys method
3. Investigate the arrows/lines in the diagram, asking the following questions:
   - What proof do you have that the cause exists?
   - What proof do you have that the cause leads to the effect?
   - Is anything else needed together with the cause for the effect to occur?
   - Is there proof that the cause is contributing to the problem I'm looking at? How much does it contribute?
4. After this questioning phase, I construct the following matrix for each and every arrow in the diagram:

Type	Question	Case-Effect arrow score
Relevancy	Do you have a proof that the cause exists?	**Yes** / No
Strength (S or W)	How strong is the proof you have that the cause leads to the effect?	H/**M**/L
Strength (S or W)	Is anything else needed together with the cause for the effect to occur?	Yes (W)/**No** (S)
Impact (D or I)	Is there a proof that the cause is contributing directly to the problem I'm looking at?	**Yes** / No
Impact (D or I)	In what way does it contributes?	**Direct** / Indirect
	Mark	

I have used High = 3 points, Medium = 2 points and Low = 1 point, and also Direct = *2, while indirect = *1.
Also, S = strong, W = Weak, D = Direct, I = Indirect.

5. Identify the routes leading to the problem/s
6. Identify the strength and direction (impact) they have (calculating the mark for each arrow)
7. Choose the best route/path to focus on, after discussing it and analyzing it
8. Act: Improve it (focus your efforts on the chosen route/path), re-create the cause-effect diagram, and go to the highest root in priority again.

In the next diagram you can see the first drawing we made in one of the projects, illustrating the initial causes identified (pointing hands).
We have identified "Only partial test planning and not full coverage" and "partial execution and low coverage".

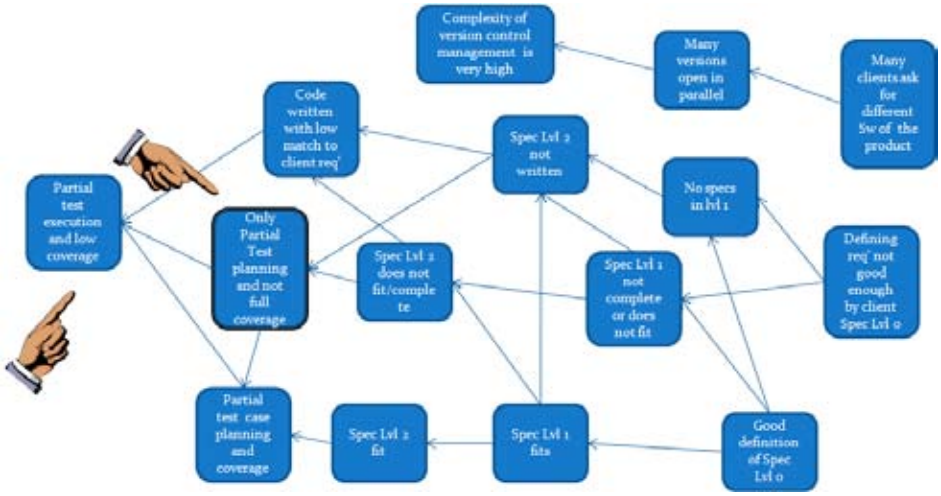


Figure-1: First draft drawing, iteration #1

Even in this diagram, we have identified a few possible improvement roots already:

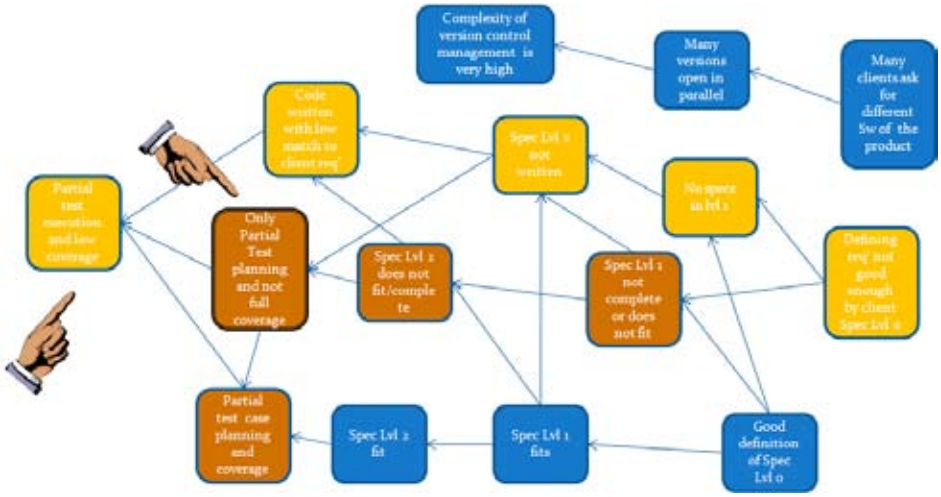


Figure-2: First draft, improvement roots identification (yellow, brown), Iteration #1

We have identified a disconnection with the upper cause-effects (blue), which is usually a sign that we do not see the whole picture yet.
We then continued discussing things, trying to identify more causes and phenomena. It is at that point that we shifted from the starting point of "partial testing being executed (not enough run)" to "poor-quality product".
It is with that focus that we continued investigating and building up the diagram. This is the next stage we reached:

Figure-3: Middle draft, identifying more causes, closing the disconnection, Iteration #3

We were now ready to investigate the arrows of the diagram and find out the scores for all of them.
We did that by, marking two signs on each arrow: strong/weak, and direct/indirect. The markings used were bold line vs. regular line for the strong/weak and straight line vs. dotted line for direct/indirect.
This was the result of the next iteration:

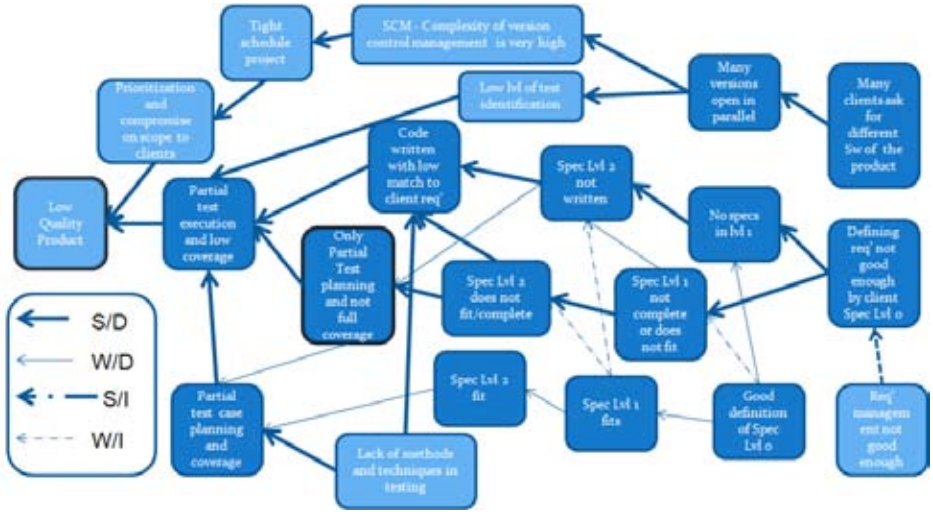


Figure-4: Final draft, identifying the strength/relevancy/direction of the arrows, iteration #4

We then went back to double-check the RCA of the routes/paths leading to the primary problem, marking the arrows with their scores.
We completed the process by circling (marked in green) the main causes that have initiated the strongest routes and are directly impacting our problem (see next figure):

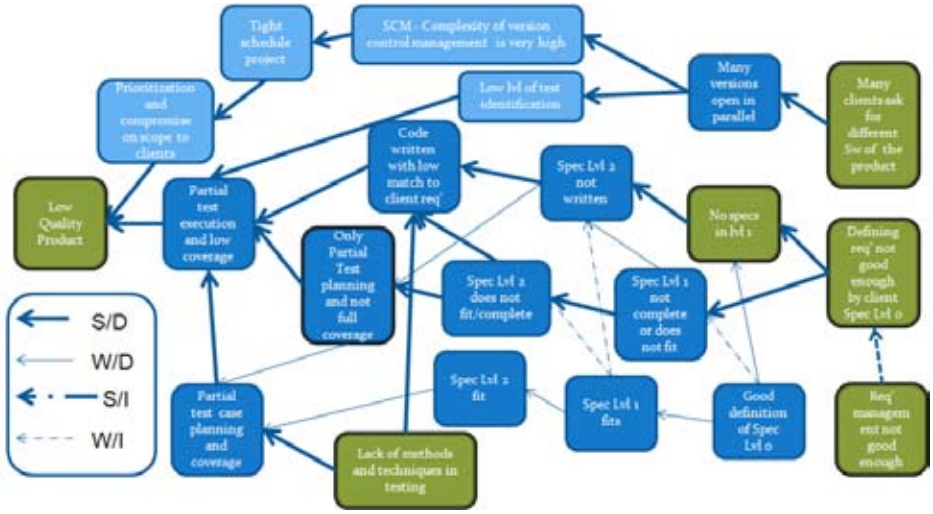


Figure-5: Final draft, identifying the root causes, and major effect, iteration #5

The next phase was to identify the possible routes/paths that lead from the root-causes to the effect. We have identified 5 such roots in that project:

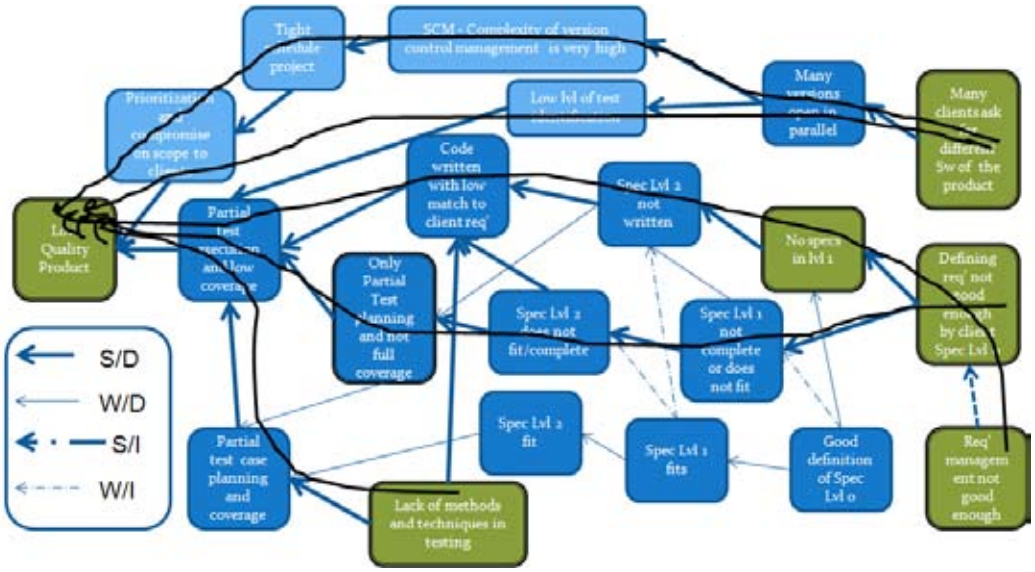


Figure-6: Final draft, Identifying possible improvement roots, iteration #6

From that point, the scores helped us realize which of the routes/paths have a higher probability to succeed and gave us more 'clear' impact on our [new] baseline statement – "poor-quality product".
After double-checking the arrow details and scores we initiated a task to build a corrective action plan for improvement.

Conclusions & Recommendations

For that project we identified 5 major Root Topics using the enhanced method and a clear process. These are explained and prioritized below:

1. Produce requirements from client definitions
2. Requirements management
3. Either 'No Spec Level 1', or 'Spec level 1 not matching requirements'
4. Lack of methods and techniques in testing for development and testing teams
5. Many clients define slightly different requirements for the SW – there are many "specials"

We defined a pragmatic plan of corrective actions with priority items, which was then realized and monitored.

Summary

A few considerations should be taken into account in order to reach a successful implementation of the enhanced method.

- Cause-effect Mapping & RCA combined can reveal real problems, leaving behind the symptoms, but we need to eliminate the problems of the model by answering the questions and scoring them on the diagram for the best route(s),
- We must deal with symptoms in the short term, otherwise…
- The questions (answers) of the enhanced method make sure that our analysis has minimal deviations, and that the route we take is a 'strong' one,
- It is a constructive process that leads to understanding, clarity, focus and the right priority setting for picking the best route for improvement implementation

I now use the method in my projects and I try to enhance it even further. Even so, there are a few open questions to be resolved in order to have this method clear and useful to everyone. I have now added some post-implementation points we will all have to consider while implementing the method.

Post-Process Observations – Next Steps

When further enhancing the model, we must think and get answers to the following questions:

- What about the junction points (inbound and outbound – see figure-7): direct impact of routes with those? Indirect? Impact on speed of performance (bottlenecks)?
- What is the ROI for this method?
- Can we validate a route before implementing it? Can we identify it to be a successful problem eliminator?
- How much of the method is context-dependant?
- Can we hook it to test process improvement methods or other key performance/area indicators?
- Other?

I am investigating these questions and looking for answers. I will welcome any cooperation in doing so.

Biography

Mr. Alon Linetzki, [MBA, B.Sc., LQA, CSM, CSA, CTFL, CTAL-TA, CTAL-TM] founder and managing director of Best-Testing (www.best-testing.com), a company which spoecializes in unique and pragmatic way of training and consulting. Mr. Linetzki has 15+ years in testing and 25+ years in IT.

Mr. Linetzki main specialized areas of consulting and training are: * test strategy & risk based testing (adding value and increasing ROI) * test management * test process improvement * defect management & analysis * building & maintaining effective and efficient testing teams.

He is a popular speaker in international testing conferences since 1995, co-founder of the ITCB - Israeli Testing Certification Board (www.itcb.org.il, 2004), and the founder and chair of SIGiST Israel (www.sigist.org.il, 2000). He is also a member of the ISTQB marketing WG.

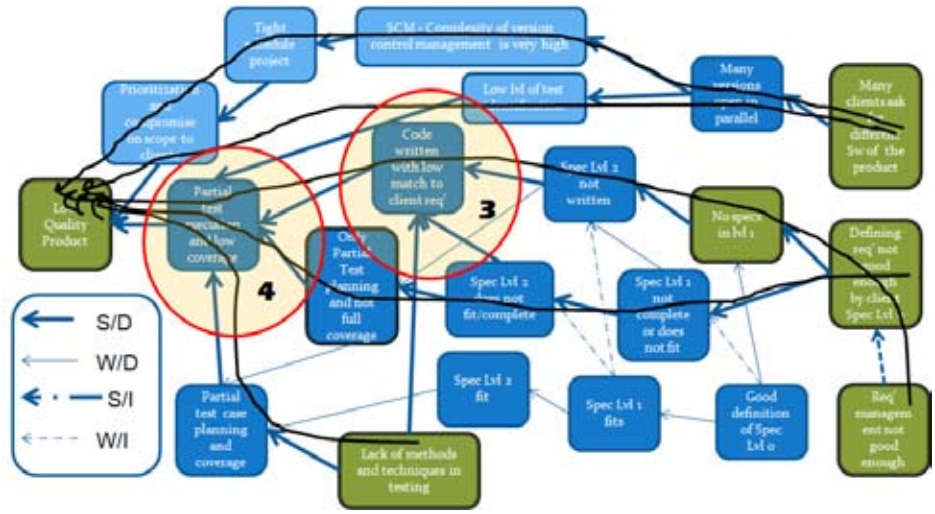
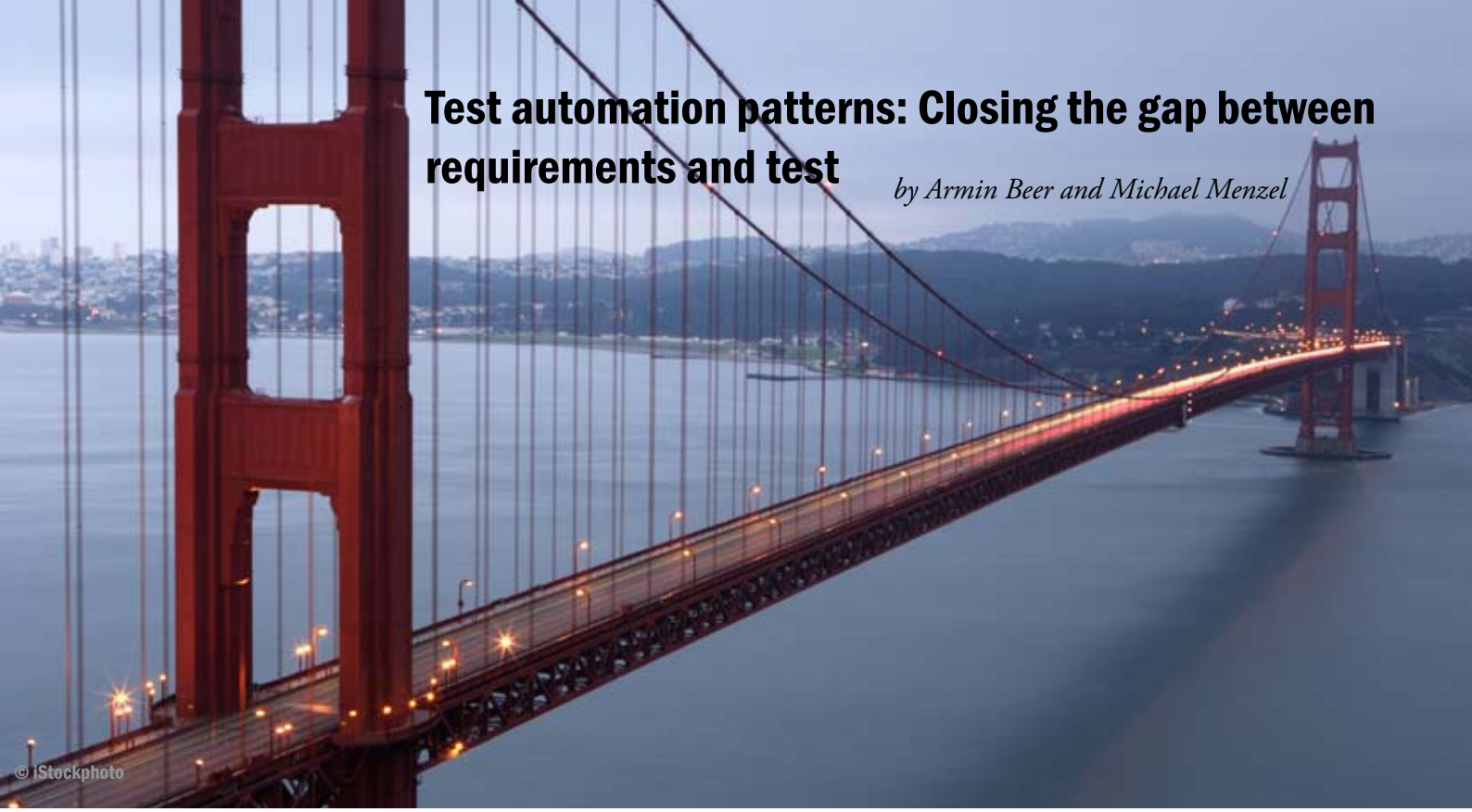


Figure-7: Final draft, inbound/outbound – circles 3 and 4

References

1. Website www.boners.com
2. Wikipedia on root cause analysis and cause effect graphing
3. Root cause analysis, cause effect mapping, and other root cause techniques, Bill Wilson, www.bill-wilson.net
4. Root Cause Analysis, Peter Abilla, 2006
5. Making Health Care Safer created for the Agency for Healthcare Research and Quality, 2001, Chapter 5
6. Website www.Moresteam.com, about 5-Whys
7. Determine the root cause: 5 whys, http://www.isixsigma.com/library/content/c020610a.asp



Test automation patterns: Closing the gap between requirements and test

by Armin Beer and Michael Menzel

Highlights:
The main goals of test automation are

- to find bugs early and fast
- to verify the main and critical functions of an application
- to test with different data sets
- to test non-functional requirements

1. Introduction

On March 27th, the new Terminal 5 of Heathrow Airport was inoperable and left thousands of people in chaos, because the baggage sorting system did not work. BAA told Computerworld UK that a computer software upgrade, conducted Monday night, was the cause of the failure. Despite testing the software, the "glitch crashed both baggage sorting machines….". British Airways had to cancel almost 450 flights, and tens of thousands of passengers suffered when their luggage remained unsorted in airport warehouses. The failure of the baggage sorting system at Terminal 5 (T5) of Heathrow Airport cost British Airways (BA) more than 16 million £ Sterling during the first week. Later the luggage had to be transported to Milan to use the baggage handling system there. The problems lasted for several weeks. (http://fin-forex.com/failure-in-heathrow-airport-work-has-turned-back-million-losses/). See also the September 2008 issue of te magazine, pp.29-33; G. Thompson: "Software testing – the future?"

What can we learn from this example?

IT plays an integral, often hidden, role in the transportation landscape. Transportation systems often contain standard IT components — clients, servers, hardware, software, networks, and so on — and are therefore subject to the kinds of flaws and breakdowns faced by corporate IT systems. The upgrade should have been tested more fully before deployment! Automated testing is essential especially if there is a tight schedule for delivery.

The main goals of test automation are

- to find bugs early and fast
- to verify the main and critical functions of an application
- to test with different data sets
- to test non-functional requirements

In the following we will introduce a maturity model and the related patterns of test automation. We will describe the patterns and discuss our observations and some arguments when applying these. The automation patterns of higher maturity and their application will be presented in one case study.

2. Frameworks for test automation

Software development in iterations requires a close cooperation between analysis, test management, development and client.(Figure 1)
Different test automation approaches for these types of projects, for example capture / replay, script programming etc. are available. Automation patterns such as capture/playback and table-driven test automation were already introduced by E. Dustin et al. in the book "Automated Software Testing" [4]. The maturity model presented here is an adaption of the model presented by Matthias Daigl at Eurostar 2002.[3] The higher the abstraction level, the higher the maturity of the automation solution.
The patterns of level 1 and 2 soon reach their limits if the tests are to be maintained and rerun over a longer period of time. Our observation is that capture/replay should only be used

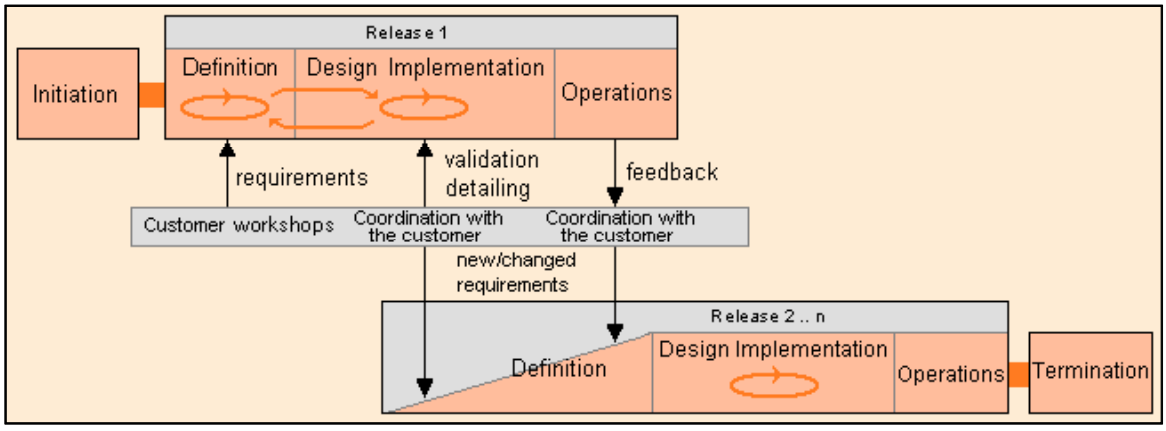


Figure 1: Iterative development

Maturity	Pattern	Remarks
Level 1	Capture / Replay	May not re-run due to limitations of the recording tool
Level 2	Script programming	Maintenance of scripts not feasible in an appropriate time frame
Level 3	Data-driven	Separation between test data and sequences; it is possible to create more tests just by adding data
Level 4	Abstract layer (wrapper)	Wrappers allow defining test cases in Excellike tables; improved maintainability of tests
Level 5	Test-case generation	Specification of test models, e.g. in UML; generation of complete test scripts; good maintainability of test cases.

Figure 2: Maturity of automation approaches

if a defect is detected by chance and the test steps have to be captured to make it reproducible for the developer. Script programming is the pattern appropriate for developers. In the course of test-driven development, developers use test frameworks, e.g. JUnit-based tools for component and integration testing. A variety of open source tools for testing at the business and control level logic (facade class API), initializing databases, coverage testing etc. enable developers to test automatically. However, in system testing, test cases are executed via the GUI. The advantage of using a test framework of level 4 maturity is that domain experts with low-levels of technical skills can also automate test cases and update them. Test case generation is the most advanced automation pattern, but needs skills in modeling e.g. in UML.

When consulting different projects, the following problems have been seen:

- An immature test management, not fulfilling the requirements of CMMI[1]: CMMI helps to integrate organizational functions, set process improvement goals and priorities, provide guidance for quality processes and gives a point of reference for appraising current processes. CMMI groups process areas into 5 maturity levels from low to high. For example, at CMMI-level 2 a configuration management must be in place and for CMMI level 3 traceability must be implemented.
- Requirements or user interfaces change too often and consequently updating takes too long.
- Bad quality of test cases, i.e. if test cases are not designed systematically [1] by testers with domain knowledge, the automated tests will not detect the defects relevant to the users.
- If testability issues are ignored, even the introduction of an automation framework may fail.
- The effort of test case maintenance is too high, because the maturity and usability of the framework does not fulfill the expectations of the testers.

We will demonstrate the practical application of the level 4 pattern in a case study in the social insurance domain.

The case study is structured into the following paragraphs:

- A short description of the project
- The test process
- The automation framework
- Results and lessons learned

3. Case study: Introduction of test automation in the social insurance domain

Project

In this case, testing of a mid-sized project for the BVA (Versicherungsanstalt öffentlicher Bediensteter) , a public social insurance company for civil servants in Austria[2] is presented. The goal was to replace the existing host application by a Java-based web application within the time frame of about one year. The project team consisted of company employees plus personnel of Siemens and other software engineering organizations. The company's employees provided the domain knowledge and the external personnel contributed the know-how about the technologies involved as well as software engineering methods and tools. The overall project team consisted of 15 to 20 people working together at the company's site. The project was split into several overlapping releases which were further structured into the phases: definition, design and implementation, and operation. Each release was put into operation once completed. The project followed an iterative development process that fostered the incorporation of results from one release into the definition of the next release.

Testing was split into unit and integration testing, system testing, and acceptance testing. Unit and integration testing was done by the developers themselves based on guidelines issued by senior developers experienced with the applied technology. Test automation was introduced during the second iteration.

A test process was already in place before the first tests were automated!

Test process

When test cases were derived from the requirements for the first release, testers found that some of the requirements were ambiguous, lacked details necessary for testing special and exceptional cases, and some were even incorrect. From the second release onward, additional test cases were designed based on the experience of the testers. All test cases were maintained in the tool SiTEMPPO (Siemens Test Execution Managing Planning and rePorting Organizer) by the domain expert testers of the insurance company. Automated regression tests for components, integration test, system tests and load and stress tests were successively implemented and maintained throughout the whole software lifecycle. The test automation framework was implemented and functional system test cases were selected from the test

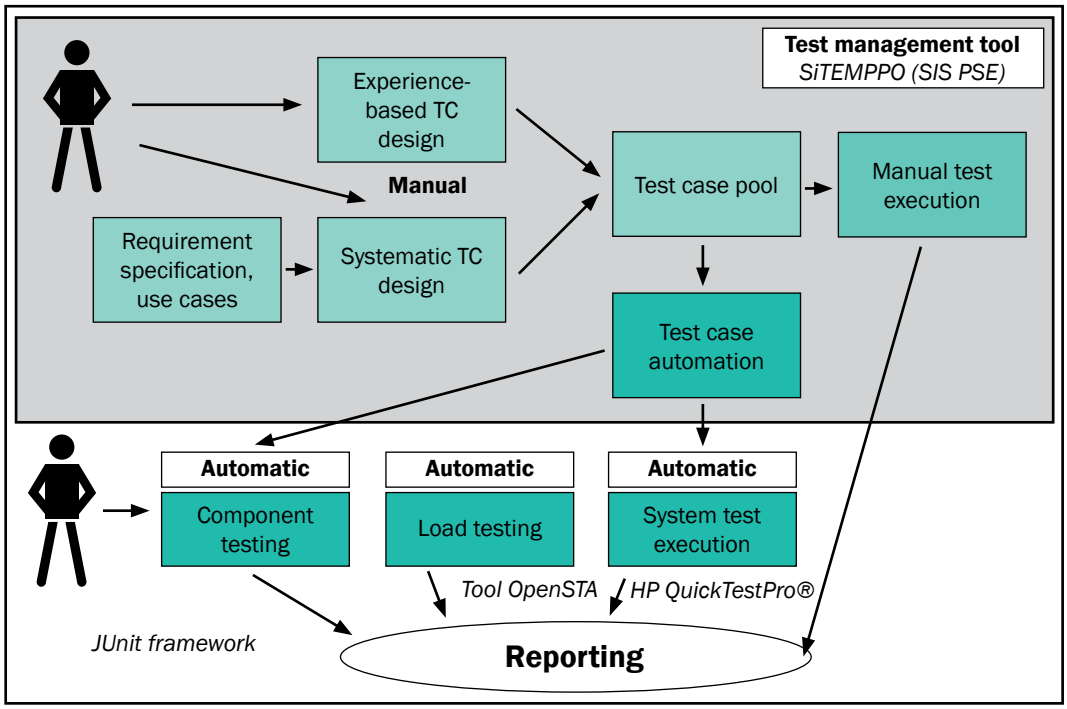


Figure 3: Test process

[1] SEI, http://www.sei.cmu.edu/cmmi/general/general.html, Retrieved Nov. 21, 2006

case pool by the testers with domain expertise when iteration 2 was developed. The selection criteria were derived from the experience gained on how the software was actually used for managing insured persons, i.e. problematic use cases as "revision" or the use cases most frequently needed by call center agents, such as "Search for insured person".
Whereas the developers followed the TDD paradigm and implemented automatic component testing, the system testers selected test cases from the test case pool for automation. Automated regression testing was performed in unit-/integration testing as well as in system testing. For unit testing a JUnit framework was used, for load testing OpenSTA and at system level HP QuickTestPro®.
Reports with the pass/fail results of component and system testing were then generated. Therefore the status and the quality of every deployment was made transparent.

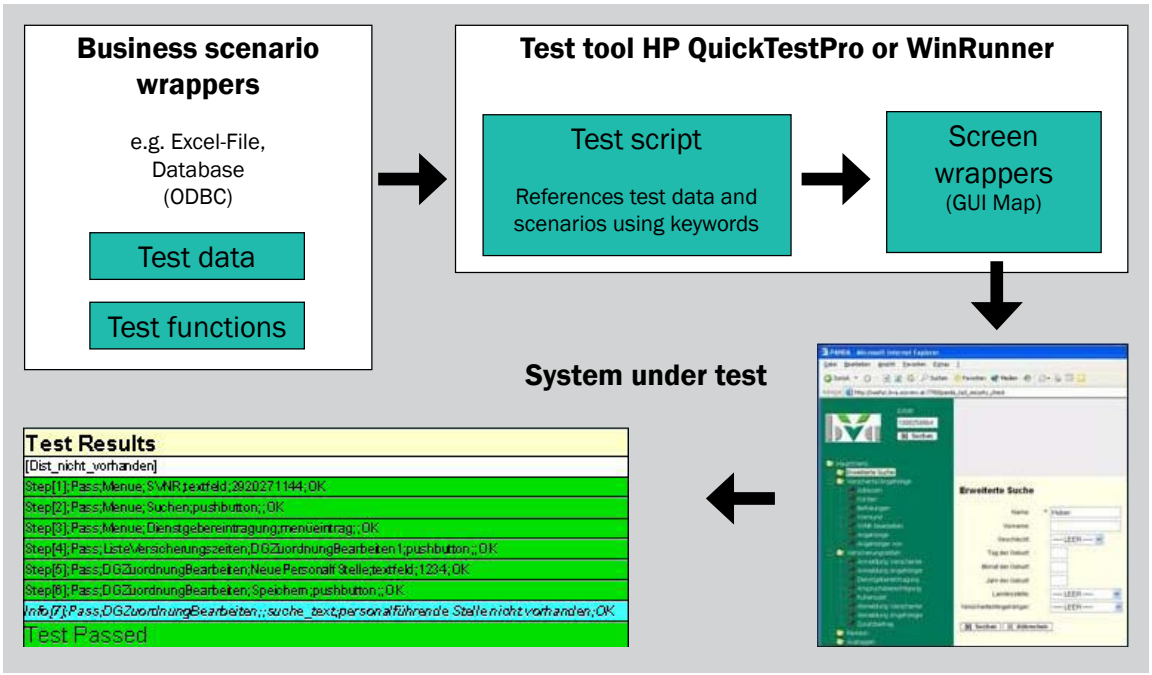


Figure 4: Level 4 test framework

Automation framework
The framework comprises a business scenario wrapper for the definition of test data and test sequences and a screen wrapper, which maps the abstract input field to the concrete GUI controls ("GUI Map"). The scripts for HP WinRunner / QuickTestPro® are generated by the framework.
The EXCEL-like front-end of the framework can easily be used by a domain expert with few technical skills.

Results and lessons learned
The advantages of this solution are: (1) Implementing re-usable test scenarios (building-block concept) improves test maintainability; (2) test step execution is traceable (one row = one step); (3) easy access to test reports via HTML. The disadvantages are: (1) high maintenance effort if test data are changed too often; (2) a new GUI map must be created if a new control is added to the GUI.

Figure 5 depicts the results of the regression tests. During the first 5 tests runs, the quality of the software was bad. With test run 6, acceptance testing was completed, and the maintenance phase started. No quality troughs occurred until run 18. Then the whole software was refactored, and 84 test cases of the subsequently conducted regression test failed. The amount of test cases increased up to 311 (about 25% of all system test cases) mainly due to new experience-based tests.
The benefit of the automatic regression tests was that every newly deployed version (about 15 in a year) could be tested automatically before the manual retesting of bugs or the testing of new features, which helped save valuable resources. The investment (294 person hours for the framework and about € 10,000 for the HP WinRunner tool) was paid off after about 6 runs (Figure 6).
After some training the domain experts were able to automate new test cases, and the automation specialist was needed only in case of specific changes in user interface layout. Automatic regression tests and regular load testing mitigated the risks of bad quality after software updates. Test automation was the main factor when it came to completing the development of P1 in time and in budget. After two years of operation in the insurance company, the user feedback on the new application is very positive.
To summarize: Project managers can make an exact quality statement before software is delivered. The test manager can assess the quality of a software version deployed, before starting with the resource-intensive manual tests. The controller can benefit from a reduction of development and maintenance costs.

4. Conclusion

Test automation plays an important role when it comes to improving defect detection and reducing quality costs. The crucial questions that have to be answered are:

- How can the risks of failure of a test automation framework be mitigated?
- Which automation frameworks are suitable for agile development and re-use?
- How efficient is test automation from the viewpoints of reduction of development costs and the quality of the product?

Special attention must be given to the maturity of the development process, e.g. in respect to requirements traceability, the test process, systematic test case design and development of skills in the team. The principal contribution of this paper is to provide a decision-making basis for a test manager faced with introducing test automation into a project. Wrapper-based test automation tools are a good solution in cases where regression tests are to be automated, for example in the social insurance domain. Test cases are more systematic – test data are generated with boundary values and cause-effect

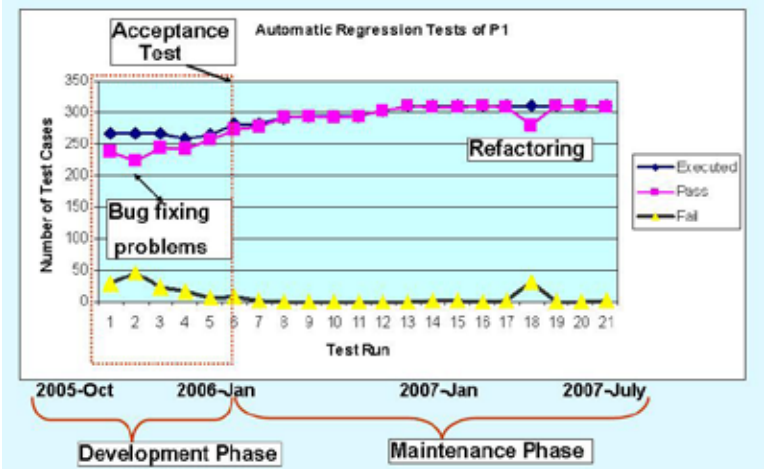


Figure 5: Level 4 test framework

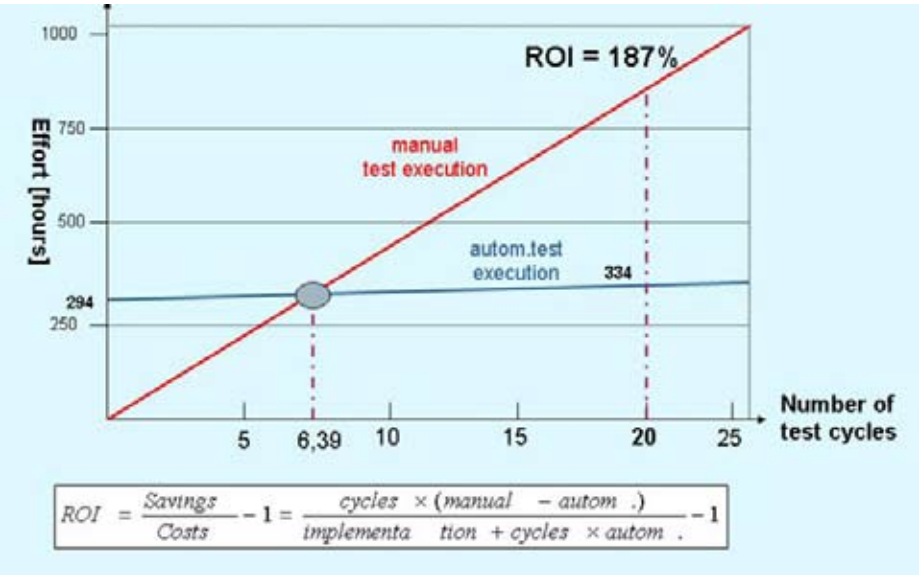


Figure 6: Return on investment in test automation in P1

analysis– and are easier to maintain.

While it is certainly sometimes difficult to obtain management approval because of the investment required for tools and skills, test automation is an important factor if the aim is to avoid delivering substandard quality. Consequently test automation is a good investment! Error costs are reduced and the risk of reappearance of already resolved defects is reduced! Also an exact quality statement becomes reality!

In conclusion, the main issues for successful test automation are symbolized by this elegant suspension bridge: Firstly, by reducing the gap in expectations between the test and development teams where tools and a test framework are in place. Secondly, using a test framework facilitates the maintenance of test cases in an iterative development process. And, last but not least, to have a mindset that the investment in test automation pays off with a significant reduction of error and maintenance costs

5. Bibliography

[1] Beer A., Mohacsi S.: "Efficient Test Data Generation for Variables with Complex Dependencies", IEEE Int. Conference on Software Testing, Verification and Validation in Lillehamer, April 2008

[2] Beer A.: "Die Einführung eines wieder verwendbaren Testframeworks in der Sozialversicherung"; Workshop: "Testing of Software – From Research to Practice" Conference on Software Engineering of the GI (Gesellschaft für Informatik); 18.-22. Februar 2008, TU München-Garching (http://se2008.in.tum.de/).

[3] Daigl M.: "Deep Impact – Reviewing 5 Years of Test Automation"; Presentation at the Eurostar in Edinburgh, Nov. 11-14, 2002,

[4] Dustin E.; Rashka J.; Paul J.: "Automated Software Testing"; Addison Wesley; ISBN-0-201-43287-0, 1999

[5] Fewster M.; Graham, D.: "Software Test Automation; Effective use of test execution"; Addison Wesley; ISBN- 0-201-33140-3, 1999

[6] Menzel M.: "Software Test Automatisierung; Leitfaden für die effiziente Einführung"; VDM Verlag Dr. Müller; ISBN-3-86550-485-8, 2006

Biography

Dipl. Ing. Armin Beer has been an independent consultant in software testing and an external contributor to Siemens IT Solution and Services (PSE) since 2008. He was a senior consultant for test management and test automation at SIS-PSE for many years. He has gained extensive testing experience in various projects, such as for safety-critical railway systems, telecommunication systems and in the social insurance domain. He is a member of the Austrian Testing Board and is involved in two ISTQB working parties, namely "Glossary" and "Expert level – test automation". He collaborates with the Technical University in Graz and the University of Applied Sciences in Vienna as a researcher and lecturer. His research in methodical testing is performed within the framework of the government-funded K-net project SoftNet and managed by Prof. Dr. Wotawa of the Technical University of Graz (www.soft-net.at).

Ing. Mag. (FH) Michael Menzel graduated in Management and Business Administration from the University of Applied Sciences in Vienna. He worked as a software tester from 1998-2002 and as a test manager in various IT projects from 2002-2005. He has been an independent consultant in software testing since 2005 (www.menzel.co.at). He is also the author of the book "Software-Testautomatisierung, Leitfaden für die effiziente Einführung"; VDM Verlag Dr. Müller, ISBN-10: 3-86550-486-8, 2006.

The Seductive and Dangerous V-Model

by James Christie

© iStockphoto

The Project Manager finishes the meeting by firing a question at the Programme Test Manager, and myself, the Project Test Manager.

"The steering board's asking questions about quality. Are we following a formal testing model?"

The Programme Test Manager doesn't hesitate.
"We always use the V Model. It's the industry standard."

The Project Manager scribbles a note. *"Good"*.

Fast forward four months, and I'm with the Programme Manager and the test analysts. It's a round table session, to boost morale and assure everyone that their hard work is appreciated.

The Programme Manager is upbeat.
"Of course you all know about the problems we've been having, but when you look at the big picture, it's not so bad. We're only about 10% behind where we should have been, and we're still on target to hit the implementation date."

This is a red rag to a bullish test manager, so I jump in.
"Yes, but that 10% is all at the expense of the testing window. We've lost half our time."

The Programme Manager doesn't take offence, and laughs.
"Oh, come on! You're not going to tell me you've not seen that before? If you're unhappy with that then maybe you're in the wrong job!".

I'm not in the mood to let it drop.
"That doesn't make it right. There's always a readiness to accept testing getting squeezed, as if the test window is contingency. If project schedules and budgets overrun, project managers can't get away with saying 'oh – that always happens' ".
He smiles and moves smoothly on, whilst the test analysts try to look deadpan, and no doubt some decide that career progression to test management isn't quite as attractive as they used to think.

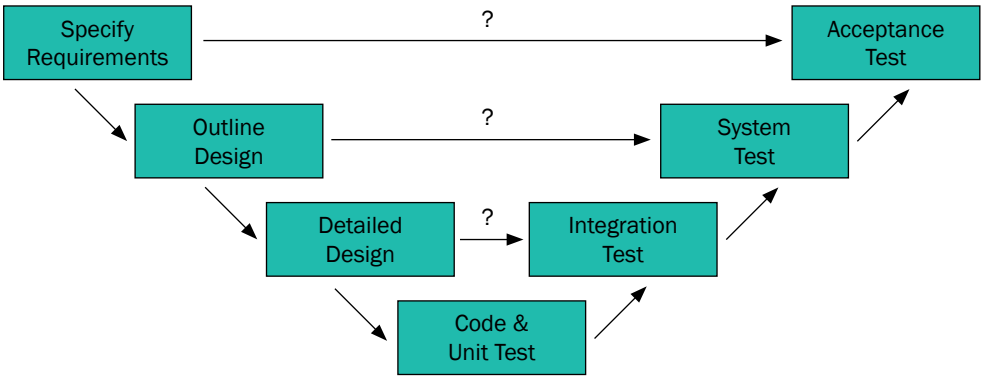
Test windows do get squeezed

The Programme Manager was quite right. That's what happens on Waterfall projects, and saying that you're using the V Model does nothing to stop that almost routine squeeze.
In "testing experience" issue 2, Dieter Arnouts [1] outlined how test management fits into different software development lifecycle models. I want to expand on the V model, its problems and what testers can do in response.
In the earlier meeting the Programme Test Manager had given an honest and truthful answer, but I wonder if he was actually wholly misleading. Yes, we were using the V Model, and unquestionably it would be the "test model" of choice for most testing professionals, in the UK at least.
However, I question whether it truly qualifies as a "test model", and whether its status as best practice, or industry standard, is deserved.
A useful model has to represent the way that testing can and should be carried out. A model must therefore be coherent, reasonably precisely defined and grounded in the realities of software development.

Coherence at the expense of precision

If the V model, as practised in the UK, has coherence it's at the expense of precision. Worldwide there are several different versions and interpretations. At one extreme is the German V-Model, [2], the official project management methodology of the German government. It's equivalent to Prince-2, but more directly relevant to software development. Unquestionably this is coherent and rigorously defined.
Of course, it's not the V Model at all in the sense that UK testers understand it. For one thing the V stands not for the familiar V shaped lifecycle model, but for "vorgehens", German for "going forwards". However, this model does promote a V shaped lifecycle, and some seem to think that this is the real V Model, in its pure form.
The US also has a government standard V Model [3], which dates back about 20 years, like its German counterpart. Its scope is narrower, being a systems development lifecycle model, but still more far more detailed and rigorous than most UK practitioners would understand by the V Model.
The understanding that most of us in the UK have of the V Model is probably based on the V Model as it's taught in the ISEB Foundation Certificate in Software Testing [4]. The syllabus merely describes the Model without providing an illustration, and does not even name the development levels on the left hand side of the V. Nor does it prescribe a set number of levels of testing. Four levels is "common", but there could be more or less on any project.
This makes sense to experienced practitioners. It's certainly coherent, in that it is intuitive. It is easy for testers to understand, but it's far from precise. Novices have no trouble understanding the model to the level required for a multiple choice exam, but they can struggle to get to grips with what exactly the V Model is.
If they search on the internet their confusion will deepen. Wikipedia is hopelessly confused, with two separate articles on the V Model. These fail to make a clear distinction between the German model [5] and the descriptive life-

cycle model [6] familiar to testers. Unsurprisingly there are many queries on internet forums asking "what ***is*** the V Model?".
There are so many variations that in practice the V Model can mean just about whatever you want it to mean.



It is interesting to do a Google images search on "V Model". One can see a huge range of images illustrating the model. All share the V shape, and all show an arrow or line linking equivalent stages in each leg. However, the nature of the link is vague and inconsistent. Is it static testing of deliverables? Simple review and sign-off of specifications? Early test planning? Even the direction of the arrow varies. The Wikipedia article on the German V Model has the arrows going in different directions in different diagrams. There is no explanation for this. It simply reflects the confusion in the profession about what exactly the V Model is.
Boiled down to its most basic form, the V Model can mean simply that test planning should start early, and test plans should be based on the equivalent level of the left hand development leg. In practice, I fear that that is all it usually does mean.
The trouble is that this really isn't a worthwhile advance on the Waterfall Model. The V Model is no more than a testing variant of the Waterfall. At best, it only mitigates some of the damaging impact that the Waterfall has on testing and quality.

Why the Waterfall is bad for testing and quality

In 1970 Royce wrote the famous paper [7] depicting the Waterfall Model. It is a strange example of how perceptive and practical advice can be distorted and misapplied. Using the analogy of a waterfall, Royce set up a straw man model to describe how software developments had been managed in the 1960s. He then demolished the model. Unfortunately posterity has credited him with inventing the model!
The problems with the Waterfall, as seen by Royce, are its inability to handle changes, mainly changes forced by test defects, and the inflexibility of the model in dealing with iteration. The implicit assumption of the Waterfall that any iteration can be safely restricted to successive stages doesn't hold. Such reworking could extend right back to the early stages of design, and is not a matter of polishing up the later steps of the previous phase.
Although Royce did not dwell on the difficulty of establishing the requirements accurately at the first pass, this problem did trouble other, later writers. They concluded that not only is it impossible in practice to define the requirements accurately before construction starts, it is wrong in principle because the process of developing the application changes the users' understanding of what is desirable and possible and thus changes the requirements. Furthermore, there is widespread agreement that the Waterfall is particularly damaging for interactive applications.
The problems of the Waterfall are therefore not because the analysts and developers have screwed up. The problems are inherent in the model. They are the result of following it properly, not badly!

The Waterfall and project management

So if the Waterfall was originally depicted as a straw man in 1970, and if it has been consistently savaged by academics and expert practitioners, why is there still debate about it? The answer lies in project management. The Waterfall was shaped by project management requirements, and it therefore facilitates project management. Along with the V model it is neater, and much easier to manage, plan and control than an iterative approach, which looks messy and unpredictable to the project manager. The Waterfall allows project managers to structure projects neatly, and it is good for ***projects not applications***!
Defenders of the model often argue that in practice the Waterfall is not applied in its pure form; that there's always some limited iteration and that there is invariably some degree of overlap between the phases. They argue that critics don't understand how it can be used effectively, and that changes are time-consuming and expensive regardless of the model followed. They dismiss the criticism that the Waterfall doesn't allow a return to an earlier stage by saying that this would be down to individual project managers being too inflexible, rather than a problem with the model.
This is naive. The problem is not solved by better project management, because project management itself has contributed towards the problem; or rather the response of rational project managers to the pressures facing them.
The symbiosis between project management and the Waterfall has meant that practitioners have frequently been compelled to employ methods that they knew were ineffective. This is most graphically illustrated by the UK government's mandated use of the PRINCE2 project management method and SSADM development methodology. These two go hand in hand.
They are not necessarily flawed, and this article does not have room for their merits and problems, but they ***are*** associated with a traditional approach such as the Waterfall.
The UK's National Audit Office stated in 2003 [8] that "*PRINCE ... fits particularly well with the 'waterfall' approach*", and that "the waterfall ... remains the preferred a*pproach for developing systems where it is very important to get the specification exactly right*".
This is current advice. The public sector tends to be more risk averse than the private sector. If auditors say an approach is "*preferred*" then it would take a bold and confident project manager to reject that advice.
This official advice is offered in spite of persistent criticism that it's never possible to define the requirements precisely in advance in the style assumed by the Waterfall, and that attempting to do so is possible only if one is prepared to steamroller the users into accepting a system that doesn't satisfy their goals.
The UK government is therefore promoting the use of a project management method partly because it fits well with a lifecycle that is fundamentally flawed because it has been shaped by project management needs rather than those of software development.
The history of the Waterfall in the USA illustrates its durability and provides a further insight into why it will survive for some time to come; its neat fit with commercial procurement practices.
The US military was the main customer for large-scale software development contracts in the 1970s and insisted on formal and rigorous development methods which effectively ruled out any alternative to the Waterfall. The reasons for this were quite explicitly to help the DoD to keep control of procurement. This bias in favour of the Waterfall lasted into the 90s, by which time the Waterfall was embedded in the very soul of the IT profession.
Even now traditional procurement practices, which fit much more comfortably with the Waterfall and the V Model, are being followed throughout the world because they facilitate control, not quality. Hence the frequent insistence on mind-numbing amounts of documentation in order to obtain approval to move on to the next stage, and the funding that goes with it. The danger is that supplier staff become fixated on the production of the material that pays their wages, rather than the real substance of the project.
It is surely significant that students of the Association of Chartered Certified Accountants are taught that the V Model is an excellent basis for planning testing. This is the only testing model that their syllabus mentions. It is the model for accountants and project managers, not developers or testers.

V for veneer?

What is seductive and damaging about the V model is that it gives credibility to the Waterfall. It gives a veneer of respectability to a process that almost guarantees shoddy quality. The most damaging aspect is perhaps the effect on usability. The V model effectively discourages user involvement in evaluating the design, and especially the interface, before the formal user acceptance testing stage. By then it is too late to make significant changes to the design. Usability problems can be dismissed as "cosmetic" and the users are pressured to accept a system that doesn't meet their needs. This is bad if it is an application for internal corporate users. It is potentially disastrous if it is a web application for customers.

None of this is new to academics or test consultants who've had to deal with usability problems. However, as Rex Black commented in his article on ISTQB Certification in issue 1 of "testing experience"; "*Common practices lag best practices by around 30 years*" [9].

Black provided a fine metaphor for this quality problem in 2002 [10]. After correctly identifying that V Model projects are driven by cost and schedule constraints, rather than quality, Black argues that the resultant fixing of the implementation date effectively locks the top of the right leg of the V in place, while the pivot point at the bottom slips further to the right, thus creating Black's "ski slope and cliff".

The project glides down the ski slope, then crashes into the "quality cliff" of the test execution stages that have been compressed into an impossible timescale.

The Waterfall may have resulted in bad systems, but its massive saving grace for companies and governments alike was that they were developed in projects that offered at least the illusion of being manageable! The Waterfall will therefore survive for a long time to come. The V Model's great attractions were that it fitted beautifully into the structure of the Waterfall, it didn't challenge that model, and it just looks right; comfortable and reassuring.

What can testers do to limit the damage?

I believe strongly that iterative development techniques must be used wherever possible. However, such techniques are beyond the scope of this article. Here I am interested only in explaining why the V Model is defective, why it has such a grip on our profession, and what testers can do to limit its potential damage.

The key question is therefore; how can we improve matters when we find we have to use it? As so often in life just asking the question is half the battle. It's crucial that testers shift their mindset from an assumption that the V Model will guide them through to a successful implementation, and instead regard it as a flawed model with a succession of mantraps that must be avoided.

Testers must first accept the counter-intuitive truth that the Waterfall and V Model only work when their precepts are violated. This won't come as any great surprise to experienced practitioners, though it is a tough lesson for novices to learn. Developers and testers may follow models and development lifecyles in theory, but often it's no more than lip service. When it comes to the crunch we all do whatever works and ditch the theory. So why not adopt techniques that work and stop pretending that the V Model does?

In particular, iteration happens anyway! Embrace it. The choice is between planning for iteration and frantically winging it, trying to squeeze in fixes and reworking of specifications.

Even the most rigid Waterfall project would allow some iteration during test execution. Try to persuade project management that test execution provides feedback about quality and risks and that it cannot be right to defer feedback. Where it is possible to get feedback early it must be taken. That means feedback to analysts and designers early enough to let them fix problems quickly and cheaply. This feedback and correction effectively introduces iteration. Acknowledging this allows us to plan for it.

Defenders of the V Model argue that that is entirely consistent with the V Model. Indeed it is. That's the point. However, what the V Model doesn't do adequately is help testers to force the issue; to provide a statement of sound practice, an effective, practical model that will guide them to a happy conclusion. It is just too vague and wishy washy.

A fundamental flaw of the V model is that it is not hooked into the construction stages in the way that its proponents blithely assume. If testing does not start early then the V Model is worthless. It is therefore important that testers agitate for the adoption of a model that honours the good intentions of the V Model, but is better suited to the realties of development and the needs of the testers.

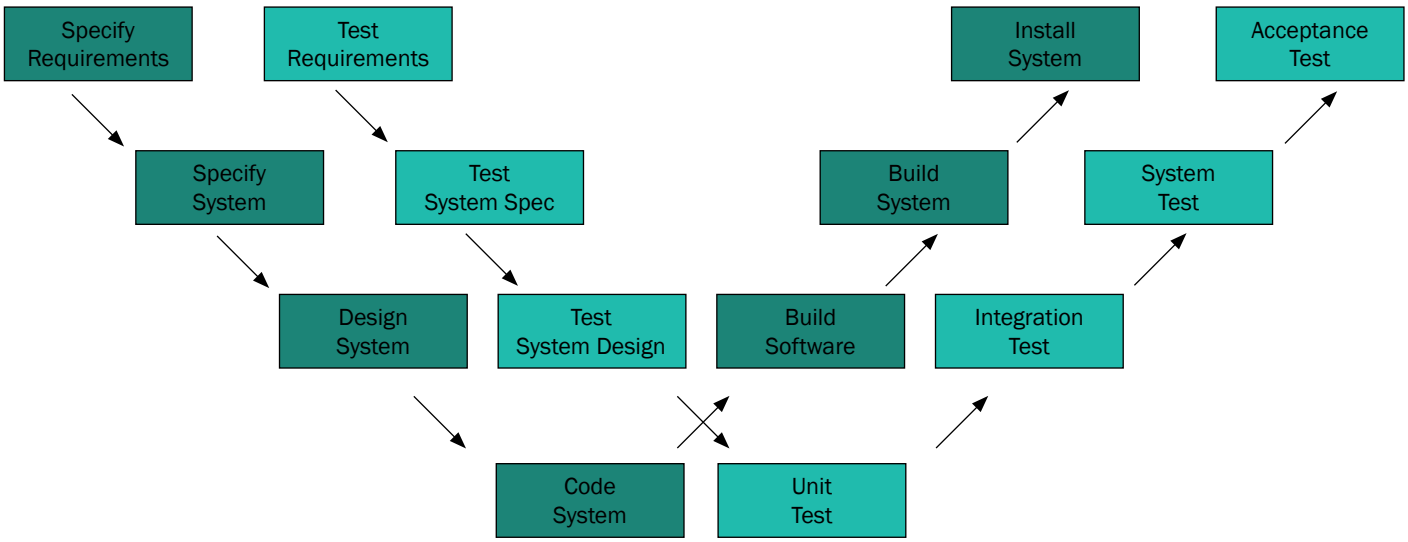
Herzlich's W Model

An interesting extension of the V Model is Paul Herzlich's W Model [11].

The W Model removes the vague and ambiguous lines linking the left and right legs of the V and replaces them with parallel testing activities, shadowing each of the development activities. As the project moves down the left leg, the testers carry out static testing (i.e. inspections and walkthroughs) of the deliverables at each stage. Ideally prototyping and early usability testing would be included to test the design of interactive systems at a time when it would be easy to solve problems. The emphasis would then switch to dynamic testing once the project moves into the integration leg.

There are several interesting aspects to the W Model. Firstly, it drops the arbitrary and unrealistic assumption that there should be a testing stage in the right leg for each development stage in the left leg. Each of the development stages has its testing shadow, within the same leg. The illustration shows a typical example where there are the same number of stages in each leg, but it's possible to vary the number and the nature of the testing stages as circumstances require without violating the principles of the model.

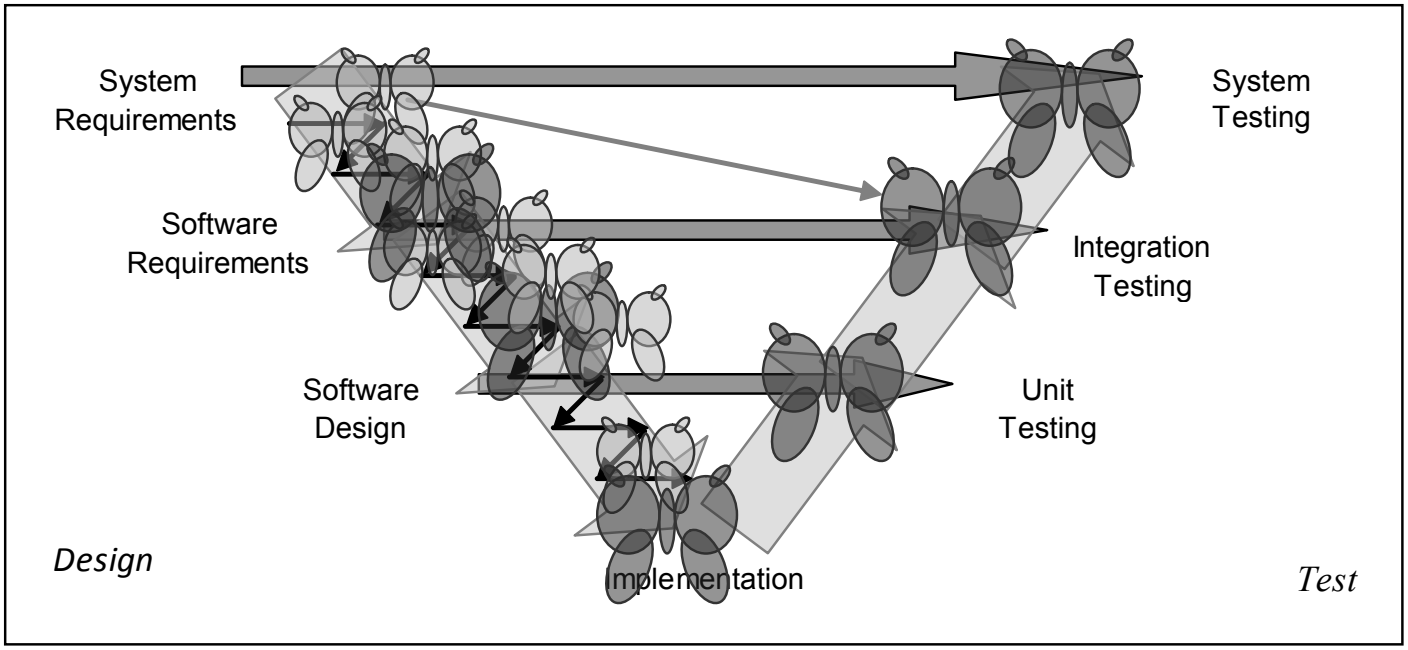
Also, it explicitly does not require the test plan for each dynamic test stage to be based on the specification produced in the twin stage on the left hand side. There is no twin stage of course, but this does address one of the undesirable by-products of a common but unthink-

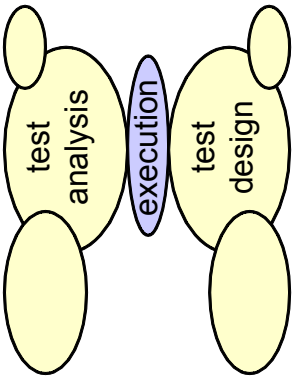


ing adoption of the V Model; a blind insistence that test plans should be generated from these equivalent documents, and only from those documents.

A crucial advantage of the W Model is that it encourages testers to define tests that can be built into the project plan, and on which development activity will be dependent, thus making it harder for test execution to be squeezed at the end of the project.

However, starting formal test execution in parallel with the start of development must not mean token reviews and sign-offs of the documentation at the end of each stage. Commonly under the V Model, and the Waterfall, test managers receive specifications with the request to review and sign off within a few days what the developers hope is a completed document. In such circumstances test managers who detect flaws can be seen as obstructive rather than constructive. Such last minute "reviews" do not count as early testing.





Morton's Butterfly Model

Another variation on the V Model is the little known Butterfly Model [12] by Stephen Morton, which shares many features of the W Model.

The butterfly metaphor is based on the idea that clusters of testing are required throughout the development to tease out information about the requirements, design and build. These micro-iterations can be structured into the familiar testing stages, and the early static testing stages envisaged by the W Model.

In this model these micro-iterations explicitly shape the development during the progression down the development leg. In essence, each micro-iteration can be represented by a butterfly; the left wing for test analysis, the right wing for specification and design, and the body is test execution, the muscle that links and drives the test, which might consist of more than one piece of analysis and design, hence the segmented wings. Sadly, this model does not seem to have been fully fleshed out, and in spite of its attractions it has almost vanished from sight.

Conclusion - the role of education

The beauty of the W and Butterfly Models is that they fully recognise the flaws of the V Model, but they can be overlaid on the V. That allows imaginative test managers to smuggle a more helpful and relevant testing model into projects committed to the V Model without giving the impression that they are doing anything radical or disruptive.

The V Model is so vague that a test manager could argue with a straight face that the essential features of the W or Butterfly are actually features of the V Model as the test manager believes it must be applied in practice. I would regard this as constructive diplomacy rather than spinning a line!

I present the W and Butterfly Models as interesting possibilities, but what really matters is that test managers understand the need to force planned iteration into the project schedule, and to hook testing activities into the project plan so that "testing early" becomes meaningful rather than a comforting and irrelevant platitude. It's possible for test managers to do any of this provided they understand the flaws of the V Model and how to improve matters. This takes us onto the matter of education.

The V model was the only "model" testers learned about when they took the old ISEB Foundation Certificate. Relatively few testers went on to take the Practitioner Certificate, which gave the more inquisitive students at least the chance to learn about other models. The Practitioner Certificate was difficult to pass and took too much time and money. For most testers it was not a high priority because there was no pressing need to pass it in order to obtain good jobs and contracts.

As a result of this, and the pressures of project management and procurement, the V model is unchallenged as the state of the art for testing in the minds of many testers, project managers and business managers.

Perhaps the introduction of the new ISEB Intermediate Certificate in Software Testing will encourage more testers to think about the different effects that different development lifecycle models have on testing. The intermediate qualification will make it easier and more attractive for testers to aim for the new Practitioner Certificate in Test Management which will introduce them to test process improvement, and possibly encourage them to find out about more sophisticated testing models like TMap.

The V model will not disappear just because practitioners become aware of its problems. However, a keen understanding of its limitations will give them a chance to anticipate these problems and produce higher quality applications.

Testers must seek to improve their knowledge and skills through formal qualifications. However, they also need to be aware of the work of the testing experts such as Rex Black, Cem Kamer, Paul Gerrard, Erik van Veenendaal and Brian Marick. Go and look for their work. And read this magazine of course!

References

1. Arnouts, D. (2008). "*Test management in different Software development life cycle models*". "Testing Experience" magazine, issue 2, June 2008.
2. IABG. "*Das V-Modell*". http://v-modell.iabg.de/index.php?option=com_docman&task=cat_view&Itemid=&gid=15&orderby=dmdate_published&ascdesc=DESC . Accessed 9th August 2008. This is in German, but there are links to English pages and a downloadable version of the full documentation in English.
3. US Dept of Transportation, Federal Highway Administration. "*Systems Engineering Guidebook for ITS*" . http://www.fhwa.dot.gov/cadiv/segb/index.htm . Accessed 9th August 2008.
4. BCS ISEB Foundation Certificate in Software Testing – Syllabus. http://www.bcs.org/server.php?show=nav.7181 . Accessed 9th August 2008.
5. Wikipedia. "*V-Model*". http://en.wikipedia.org/wiki/V-Modell . Accessed 9th August 2008.
6. Wikipedia. "*V-Model (software development)*". http://en.wikipedia.org/wiki/V-Model_(software_development) . Accessed 9th August 2008.
7. Royce, W. (1970). "*Managing the Development of Large Software Systems*", IEEE Wescon, August 1970.
8. National Audit Office. (2003)."*Review of System Development – Overview*". www.nao.org.uk/intosai/edp/Audit%20Guides/Review%20of%20systems%20development%20-%20overview.pdf . Accessed 26th July 2008.
9. Black, R. (2008). "*ISTQB Certification: Why You Need It and How to Get It*" . "Testing Experience" magazine, issue 1, March 2008.
10. Black, R. (2002). "*Managing the Testing Process*", p415. Wiley 2002 .
11. Herzlich, P. (1993). "*The Politics of Testing*". Proceedings of 1st EuroSTAR conference, London, Oct. 25-28, 1993.
12. Morton, S. (2001). "*The Butterfly Model for Test Development*". Sticky Minds website. http://stickyminds.com/getfile.asp?ot=XML&id=2618&fn=XUS441382file1%2Edoc . Accessed 9th August 2008.

Biography

James lives in Perth in Scotland . He is currently working as a consultant through his own company, Claro Testing Ltd (www.clarotesting.com).

James has 24 years commercial IT experience, mainly in financial services, with the General Accident insurance company and IBM (working with a range of blue-chip clients) throughout the UK and also in Finland.

His experience covers test management (the full life-cycle from setting the strategy, writing the test plans, supervising execution, through to implementation), information security management, project management, IT audit and systems analysis.

In 2007 he successfully completed an MSc in IT. His specialism was "Integrating usability testing with formal software testing models". He is particularly interested in the improvement of testing processes and how the quality of applications can be improved by incorporating usability engineering and testing techniques.

James is a Chartered IT Professional and a professional member of the British Computer Society (MBCS). He holds the ISEB Practitioner Certificate in Software Testing and is a member of the Usability Professionals Association.

Robot Framework for Test Automation

by Marcin Michalak and Pekka Laukkanen

Introduction

Robot Framework [1] is a Python-based keyword-driven test automation framework for acceptance level testing and acceptance test-driven development (ATDD). It has an easy-to-use tabular syntax for creating test cases. Robot testing capabilities can be extended by test libraries implemented either in Python or Java. Users can also create new keywords from existing ones using the same simple syntax that is used for creating test cases.Test case results are stored in XML or easily readable HTML files.

Robot Framework is an outcome of the Master's Thesis research, and has since been developed as an internal tool at NokiaSiemensNetworks. This spring NSN has agreed to make it an open-source tool and it can since be widely used.

This article will present the basics of Robot, focusing on writing, executing and analyzing test cases. All sections are illustrated by a real-life example of a simple web application.

Preparing test cases

In Robot, test cases are written in a tabular format. They can be saved either in HTML or tab-separated value (TSV) files. These files can be created in any HTML/TSV editor. For users' convenience, there is also a Robot Framework IDE [2], which was specially developed to make test case editing simple.

Let's presume, we have a simple web application that we need to test. A 'Valid login' test case is presented in Figure 1.

Keyword	Action	Argument	Argument
Open Login Page	Open Browser	http://host/login.html	
	Title Should Be	Login Page	
Title Should Start With	[Arguments]	${expected}	
	${title} =	Get Title	
	Should Start With	${title}	${expected}

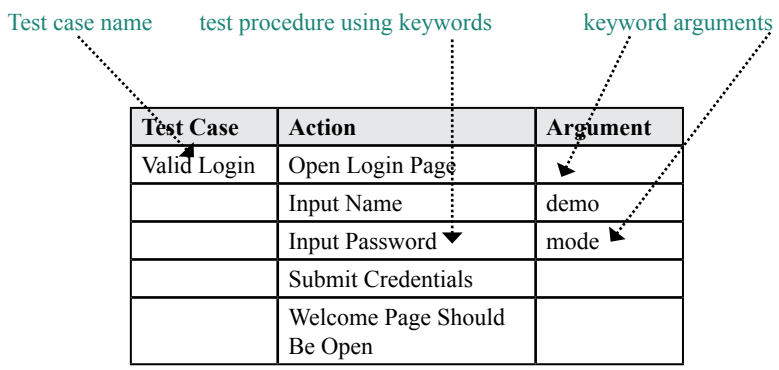
Figure 2. User keyword implementation

All entries in the 'Action' column are keywords (functions/procedures). These keywords can be implemented inside the 'Keyword' table, using other user-defined keywords or a wide range of built-in functions. For more complex functionalities, it is possible to use Python or Java-based libraries. Some already exist (OperatingSystem, Selenium, Telnet, SSH, and Swing libraries), and others are under development.

Implementing own keywords in Robot is very simple. All we need to do is to describe the actions with arguments in the 'keyword' table, as in the example in Figure 2. Simple and intuitive syntax allows for rapid keyword development, even by testers without programming experience.

'Open Browser' and 'Title should be' are internal keywords from the Robot web-testing library using Selenium.

We can have test cases which have a unique procedure, or test cases that are similar to each other, where the procedure is always the same and only the input data changes. Robot also supports this data-driven approach. All one needs to do is to define a keyword which will take the input data and prepare a table with test cases.



Test Case	Action	Argument
Valid Login	Open Login Page	
	Input Name	demo
	Input Password	mode
	Submit Credentials	
	Welcome Page Should Be Open	

Figure 1. Test case table for valid login

Test Case	Action	User Name	Password
Invalid User Name	Login With Invalid Credentials Should Fail	invalid	mode
Invalid Password	Login With Invalid Credentials Should Fail	demo	invalid
Invalid User Name And Password	Login With Invalid Credentials Should Fail	invalid	whatever
Empty User Name	Login With Invalid Credentials Should Fail		mode
Empty Password	Login With Invalid Credentials Should Fail	demo	\
Empty User Name And Password	Login With Invalid Credentials Should Fail		\

Figure 3. Test cases using data-driven approach (single \ means an empty string)

Test case organization

There are several ways of managing test cases in Robot. Test cases are organized into test suites, which are sets of test cases, taken either from single or multiple files. The simplest example of a test suite is a single HTML file containing multiple test cases.
Another key feature of Robot is test case tagging. Each tag (e.g. critical, nightly, time-consuming, quick) assigns a test case to a separate set. When executing, tests can be selected based on tags, like "critical & nightly". Those tags are also included in the execution statistics later.

The tester can set many options from the command line: select the test cases by name or tag, set variables, execution order. Especially the last option is very interesting: it is possible to execute test cases in a random order, improving the probability to find defects.

Logs and reports

Another strong point of Robot are logs and reports. They are presented in a very readable format in HTML files. Reading logs is very intuitive: they are presented in a tabular/tree format, where green/red colors are used to mark passed and failed.
Reports give an overview of the test cases executed: their names, execution time, duration, and final result. Logs provide more information on the execution, including all the keywords. One can easily trace the execution flow, which makes logs a perfect help for debugging.
(see Figure 6)

Statistics by Tag	Total	Pass	Fail	Graph
regression	10	4	6	
smoke	4	4	0	

Figure 4. Statistics per tag in report

Executing test cases

When the test cases are written and ready, they can be executed using provided Python scripts. Robot Framework is system and platform independent - test cases can be executed on any platform where Python is available: be it Windows, Linux, Unix or Mac.
The execution of test cases is done in the following steps:

- gathering test cases, reading and setting variables
- executing all actions for every test case
- providing global statistics (test cases passed/failed)
- writing the log in xml format
- generating report and log in html format.

The execution script writes the basic information to the console, as in the example below:

```
==============================================================
Example test suite
==============================================================
First test :: Possible documentation is here | PASS |
--------------------------------------------------------------
Second test | FAIL |
Error message is displayed here
==============================================================
Example test suite | FAIL |
2 critical tests, 1 passed, 1 failed
2 tests total, 1 passed, 1 failed
==============================================================
Output: /path/to/output.xml
Report: /path/to/report.html
Log: /path/to/log.html
```

Figure 5. Execution of test cases - command line output

After executing the test cases, Robot saves the log into an XML file. This XML file can later be converted into an HTML file or processed further according to the tester's needs.
In the example presented in Figure 7, the test case failed because the opened page was different from the one expected. We can see all the actions executed up to the failing action, including the fail message. Initially, all the keywords and test cases executed successfully are not expanded – only the top information is shown. If we need more details, we can expand the tree using the (+) button.
(see Figure 7)

Tested application interface

There are three main domains where Robot can be applied: command-line, web and GUI testing. Other interfaces can be tested using self-developed libraries.
For command-line testing, we have two useful libraries: OperatingSystem and SSHLibrary. OperatingSystem allows us to execute system commands, manipulate files, check files, read/set environment and manage processes. SSHLibrary allows us to connect to a remote host, execute commands and transfer files. Using those two libraries, we can create a powerful set of keywords for testing our application through the command line, whether it is installed locally or on a remote host. The example below shows a simple test case using command-line ping. It would also be possible to create a keyword 'Host should respond to ping' and just use it with a certain hostname/ip in the test case.

(see Figure 8)

The second interface type is the web using real browsers. In this case, we have a Robot Selenium library [3], which is a wrapper around Selenium Web application testing system [4].
The Robot Selenium library contains keywords to open a web page in the browser, input text, click button, submit, etc. It contains a lot of testing functions, like 'Location should be', 'Page should contain text field', 'List selection should be' and many others. A sample keyword implemented using the Selenium library is presented in Figure 2 - all presented web application tests are running with Selenium.

The third interface supported by Robot Framework is GUI. Currently the Swing GUI library exists. It provides keywords such as 'Dialog should be open', 'Push button', 'Menu item should be selected', 'Start application', 'Get selected table cell value', 'Close window' and many others. Figure 8 presents a sample user keyword implemented using Swing library.

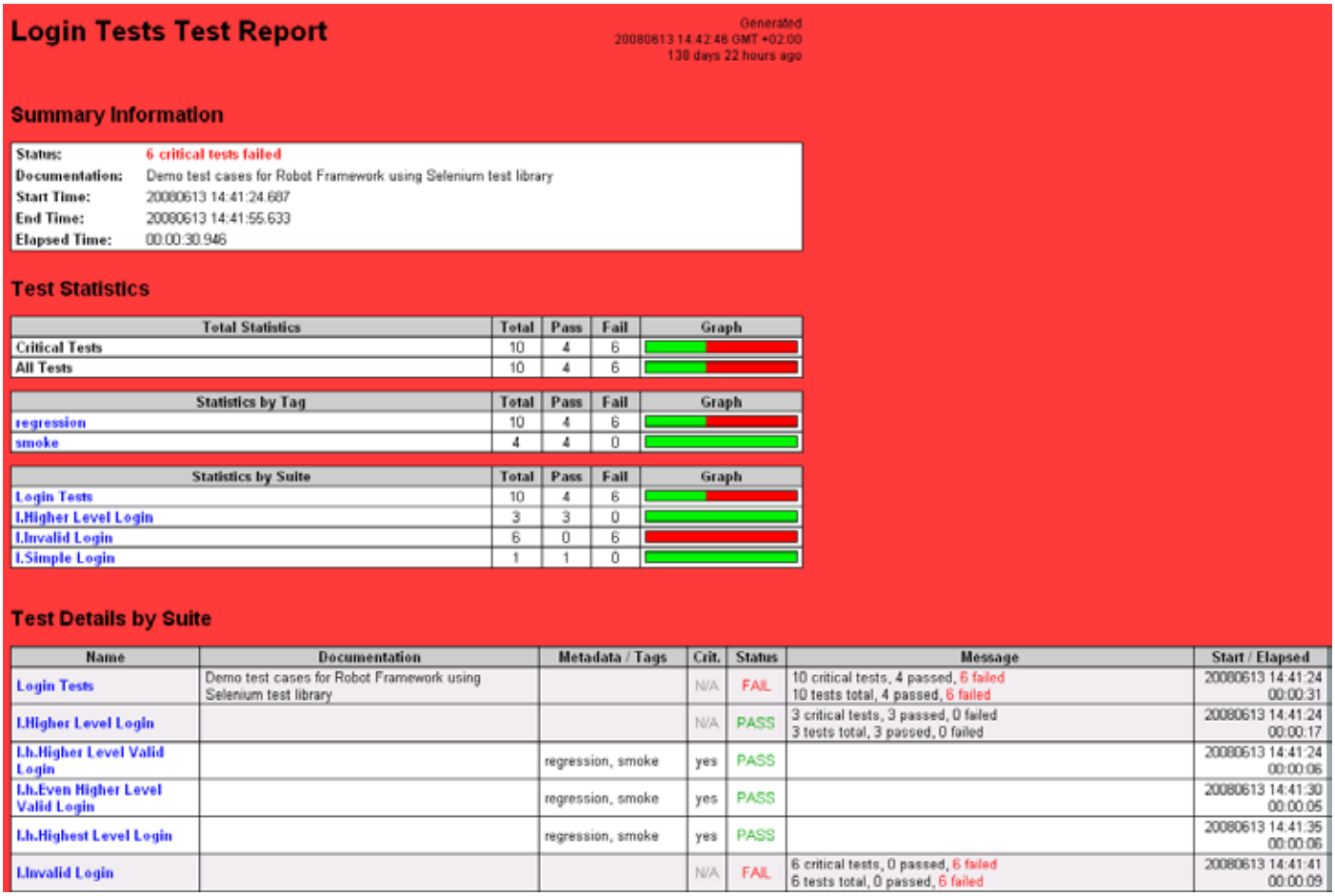
(see Figure 9)

Other interfaces can also be tested - it is possible to access the application using native libraries or standard/non-standard protocols. In this case, a wrapper must be written in Python or Java that will make user-level functions accessible to Robot. The user guide contains such examples.

Being Agile using Robot

Robot Framework is very helpful if projects are run in Agile mode. It supports test-driven development, where test cases are written before the implementation is ready. Since Robot syntax is similar to the natural language, the text of the test case may directly become its documentation. All written test cases can be executed e.g. every night - the challenge for the team is to make them all turn green.

(see Figure 10)



Login Tests Test Report

Generated
20080613 14:42:46 GMT +02:00
130 days 22 hours ago

Summary Information

Status:	6 critical tests failed
Documentation:	Demo test cases for Robot Framework using Selenium test library
Start Time:	20080613 14:41:24.687
End Time:	20080613 14:41:55.633
Elapsed Time:	00:00:30.946

Test Statistics

Total Statistics	Total	Pass	Fail	Graph
Critical Tests	10	4	6	
All Tests	10	4	6	

Statistics by Tag	Total	Pass	Fail	Graph
regression	10	4	6	
smoke	4	4	0	

Statistics by Suite	Total	Pass	Fail	Graph
Login Tests	10	4	6	
I.Higher Level Login	3	3	0	
I.Invalid Login	6	0	6	
I.Simple Login	1	1	0	

Test Details by Suite

Name	Documentation	Metadata / Tags	Crit.	Status	Message	Start / Elapsed
Login Tests	Demo test cases for Robot Framework using Selenium test library		N/A	FAIL	10 critical tests, 4 passed, 6 failed 10 tests total, 4 passed, 6 failed	20080613 14:41:24 00:00:31
I.Higher Level Login			N/A	PASS	3 critical tests, 3 passed, 0 failed 3 tests total, 3 passed, 0 failed	20080613 14:41:24 00:00:17
I.h.Higher Level Valid Login		regression, smoke	yes	PASS		20080613 14:41:24 00:00:06
I.h.Even Higher Level Valid Login		regression, smoke	yes	PASS		20080613 14:41:30 00:00:05
I.h.Highest Level Login		regression, smoke	yes	PASS		20080613 14:41:35 00:00:06
I.Invalid Login			N/A	FAIL	6 critical tests, 0 passed, 6 failed 6 tests total, 0 passed, 6 failed	20080613 14:41:41 00:00:09

Figure 6. Execution report, including statistics and fail messages

TEST CASE: **Invalid Password**

Full Name: Login Tests.Invalid Login.Invalid Password
Critical: yes
Tags: regression
Start Time: 20080613 14:41:46.972
End Time: 20080613 14:41:47.610
Elapsed Time: 00:00:00.638
Status: FAIL
Message: Location should have been 'invalid' but was 'http://localhost:7272/error.html'

KEYWORD: **Login With Invalid Credentials Should Fail** demo, invalid

Start Time: 20080613 14:41:46.973
End Time: 20080613 14:41:47.362
Elapsed Time: 00:00:00.389

KEYWORD: **resource.Input Username** ${username}

KEYWORD: **resource.Input Password** ${password}

KEYWORD: **resource.Click Login Button**

KEYWORD: **SeleniumLibrary.Location Should Be** ${ERROR URL}

Documentation: Verifies the current URL is exactly 'url'.
Start Time: 20080613 14:41:47.308
End Time: 20080613 14:41:47.362
Elapsed Time: 00:00:00.054

```
14:41:47.360  INFO  Verifying current location is 'invalid'.
14:41:47.361  FAIL  Location should have been 'invalid' but was 'http://localhost:7272/error.html'
14:41:47.361  INFO  Traceback (most recent call last):
                    File "/usr/lib/python2.5/site-packages/SeleniumLibrary/__init__.py", line 140, in location_should_be
                      asserts.assert_equal(url, actual, msg, False)
                    File "/usr/lib/python2.5/site-packages/robot/utils/asserts.py", line 176, in fail_unless_equal
                      _report_unequality_failure(first, second, msg, values, '!=')
                    File "/usr/lib/python2.5/site-packages/robot/utils/asserts.py", line 239, in _report_unequality_failure
                      _report_failure(msg)
                    File "/usr/lib/python2.5/site-packages/robot/utils/asserts.py", line 225, in _report_failure
                      raise AssertionError(msg)
```

TEARDOWN: **Go To Login Page**

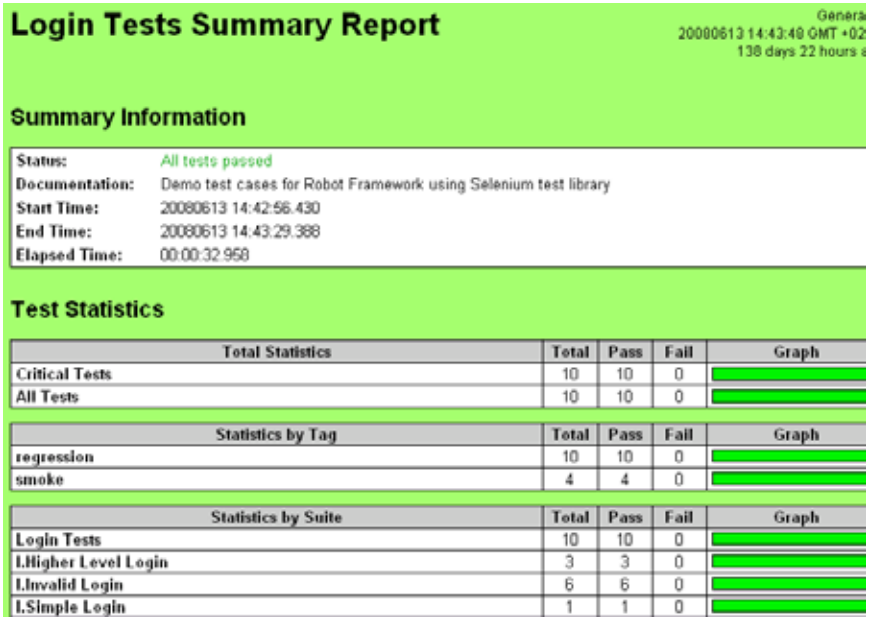
Figure 7. Failed test case log, expanded on the failed keyword

Test Case	Action	Argument	Argument	Argument
Test case Ping	[Timeout]	30s		
	${rc}	${output}	Run and Return RC and output	ping www.google.com
	Should be equal as integers	${rc}	0	Ping failed: ${output}
	Should not contain	${output}	Received = 0	Received 0 packets - host not responding

Figure 8. Ping test case using command-line

Keyword	Action	Argument	Argument
Add Test Data To Registration List	Insert Into Text Field	name_textfield	John Doe
	Insert Into Text Field	email_textfield	john.doe@acme.com
	Push Button	add_button	

Figure 9. User keyword using Swing library



Login Tests Summary Report

Summary Information

Status:	All tests passed
Documentation:	Demo test cases for Robot Framework using Selenium test library
Start Time:	20080613 14:42:56.430
End Time:	20080613 14:43:29.388
Elapsed Time:	00:00:32.958

Test Statistics

Total Statistics	Total	Pass	Fail	Graph
Critical Tests	10	10	0	
All Tests	10	10	0	

Statistics by Tag	Total	Pass	Fail	Graph
regression	10	10	0	
smoke	4	4	0	

Statistics by Suite	Total	Pass	Fail	Graph
Login Tests	10	10	0	
Login.Higher Level Login	3	3	0	
Login.Invalid Login	6	6	0	
Login.Simple Login	1	1	0	

Figure 10. Team challenge - make all testcases passed

Summary and future

Robot is a very simple and easy-to-use, flexible, acceptance-level test automation framework. It is system and platform independent and is extensible through Python and Java modules. Test cases can be written in a simple, tabular format, even by people without programming experience, using built-in or self-developed keywords. The logs and reports produced are easily readable and provide a perfect source for test case debugging. Currently Robot is widely used and appreciated within NokiaSiemensNetworks. Since becoming an OpenSource project this spring, it has many users outside the company. It is continuously developed and its base of keywords is growing. We strongly encourage users to create and share their own libraries and experiences.

For more information, please refer to the Robot Framework Main Page, including User's Guide, test case samples and Forum Board.

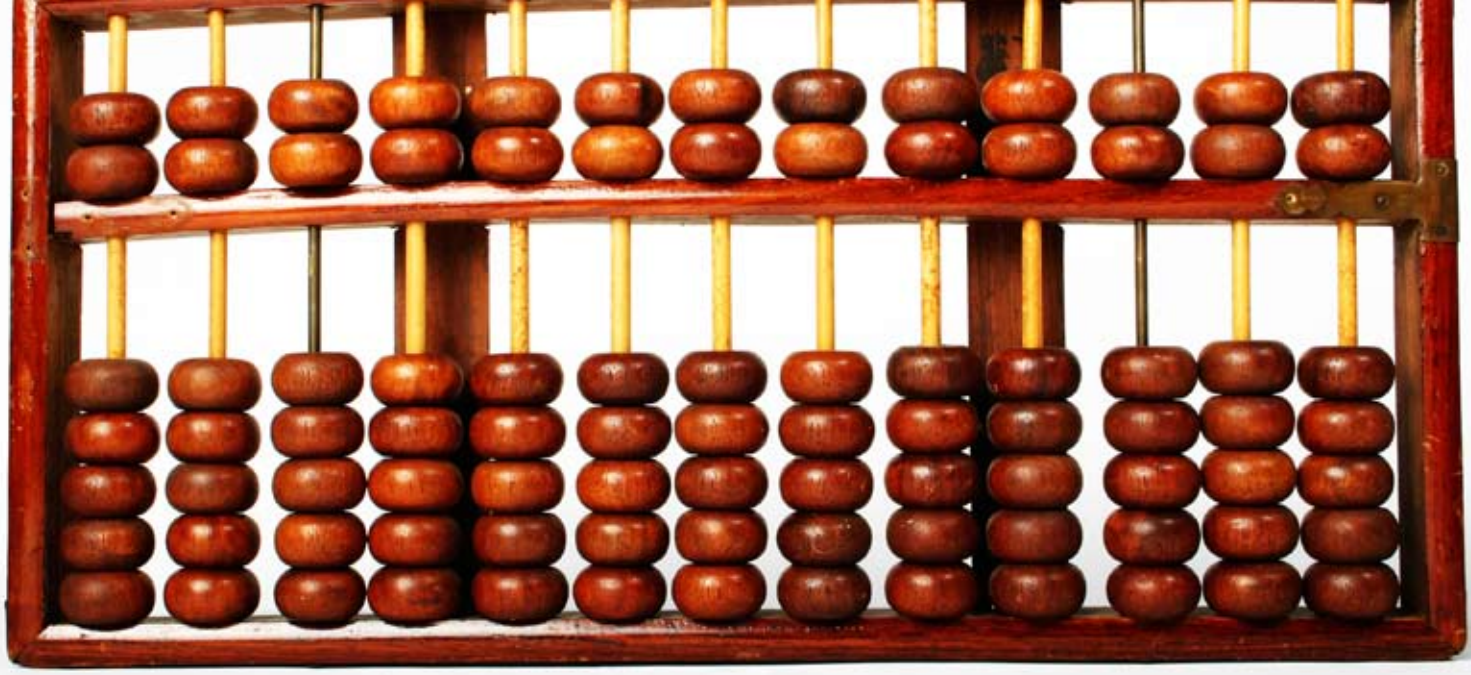
Robot Framework Links

[1] http://robotframework.org: Robot Framework Main Page
[2] http://code.google.com/p/robotframework-ride - Robot Framework IDE
[3] http://robotframework-seleniumlibrary.googlecode.com - Robot Selenium Library
[4] http://selenium.openqa.org/ - Selenium Web Application Testing System

Biography

Marcin Michalak
Marcin has received the Master of Science in Computer Engineering title from the University of Mining and Metallurgy in Krakow, Poland. From 2001 until 2007 he worked as research engineer in Belgium and Switzerland, doing research on IPv6, QoS and Wireless Networks. In 2007, he joined NokiaSiemensNetworks in Wroclaw, Poland, working as a Test Automation Engineer. His work focuses on developing automated tests and libraries for Netact OSS and providing training and support in test automation. He is also an ISTQB-certified tester. Marcin's interests include test automation tools and technologies, as well as next-generation networks and communication solutions.

Pekka Laukkanen
Pekka holds a Master of Science degree from Helsinki University of Technology and has been involved in test automation activities since 2000. He has created test automation tools and frameworks using Java, Perl, shell scripts and, nowadays, mainly Python. Robot Framework originated from Pekka's master's thesis on large-scale test automation frameworks.
Following his interest in Agile software development, Pekka became a Certified Scrum Master in 2006. He also follows and supports many open-source projects.
Currently he is working as a free consultant for his own company Eliga Oy.



© iStockphoto

The cost of automation. A method to resolve the convenience of test automation

by Albert Farré

Introduction

Testing is a substantial part of the software life cycle and in cases where quality is essential, the testing phase becomes critical. This is the case for clinical software or software for the aerospace market. However, testing may become inefficient or boring after several cycles of fixing and re-testing. In addition, it usually takes a long time to assure the required level of quality.

By automating testing, it is expected that the length of the testing phase is reduced, that a higher efficiency is achieved through repeatability, and finally, that the burden placed on software testers is reduced to a degree that will improve their awareness and, eventually, their productivity.

In spite of these benefits, test automation is not a rose garden and pitfalls abound. Indeed, many risks are involved when automating a test and costs may increase beyond expectations if unnecessary effort is allocated to automation. Also, bad automation may lead to defects slipping through and therefore stalling the test effectiveness. There is a possibility that you end up with an automated test suite that requires so much test reworking for each new version that the tester just cannot cope with any other tasks while updating test procedures.

This brings stakeholders to the crossroads between automating and not automating. Several authors have given general (good) guidelines on the topic [1],[2],[3]. In most cases, guidelines require the presence of an expert's assessment, but at the same time, they provide the expert with general advice rather than precise methods. Thus, it is clear there is a need for a method to support the decision to automate test suites.

Objective: Define a method

In order to define a method, the first step is to establish a clear goal for it. As commented in the introduction, it must be judged whether a test suite is to be automated or not.

However, it would be unwise to apply a decision Automate/Do-not-automate for the whole test suite. The reason is that complete automation is almost impossible to reach for most projects without incurring extreme costs. On the other hand, discarding automation due to difficulties in some very specific areas also seems inappropriate. It may be the case that the rest of the test suite fits into automated procedures like a glove.

Instead, it is probably wiser to decide how far the test automation should go rather than an all-or-nothing approach. As a consequence, the following possible decisions (figure 1) can be considered:

- Complete automation (at least theoretically)
- Limited functional automation (only specific functionality areas)
- Test Case (TC) partial automation (only a specific part of each
- TC)
- Limited functional and TC Partial automation (only a specific part of each TC and constrained to specific functionality areas)
- No automation at all

Furthermore, it is likely that automation ends up being possible, but not for free. Indeed, test automation always has a cost. Since these costs may be acceptable in some projects due to their ROI and not in others, it is worthwhile to give cost estimations. Giving qualitative and quantitative estimations will provide stakeholders with the necessary arguments to take a decision.

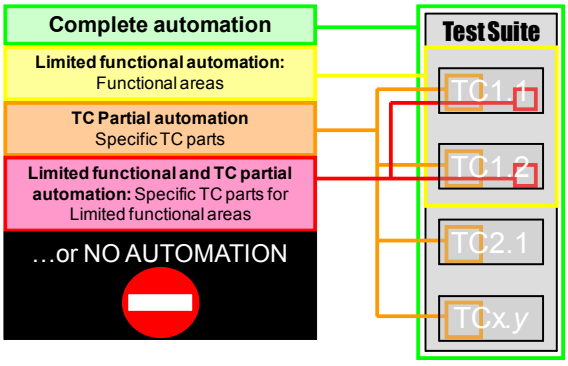
Therefore, the goal of the method is to evaluate and distinguish those areas and TC parts which are apt to be tested automatically. As a secondary objective, it focuses on estimating the cost of automation, both qualitatively and quantitatively.

The method in a few words

The method involves the evaluation of each TC by an expert. Afterwards, according to this evaluation, the quantitative cost is calculated and a qualitative cost is associated to the TC. Finally, quantitative and qualitative results are gathered and a decision is taken by a group of stakeholders. The decision is based on the cost estimates and the automation goals.

Part 0. Automation Goal

It may seem obvious, but the very first thing to do is to set an automation goal. The goal should clearly describe what is to be achieved through automation.



Goals can be related to a reduction in execution time, or to an increase in coverage within a constrained execution time, or to allow multiple executions of the same test for different platforms without increasing human resources. The goal may also be expressed as a statement of intentions. It may be in terms of ROI (Return on Investment).

Whatever the goal and whichever way it is expressed, it is required to guide the evaluation and decision making process. As a matter of fact, the automation goal shall be explicitly used by the stakeholders as the main argument to tip the scales in disputed cases. At the same time, the expert evaluation shall be driven by the automation goal.

Part I. Test Case Evaluation

As mentioned above, the process starts with the evaluation of TCs by an expert. This evaluation is roughly described as an inspection of each test step, whereby each test step is classified in a category. Decisions may be based on quantitative results or they may consider qualitative aspects, or both. As a consequence, it is advisable for the expert to provide assessment in both ways.

EVALUATION PRINCIPLES

In order to assess the automation suitability for each case, the expert should take into account a few more factors. For starters, the expert usually takes part in the stakeholders meeting to provide better insight. Indeed, the same expert may end up automating the TC. But this doesn't always have to be the case. Thus, any comment added to the assessment may be useful in later stages if the TC is automated by other engineers.

In addition, evaluation shall take into account the tools that are suitable for automating the TC. Different tools may have different degrees of compatibility with the application, and therefore, the choice of one tool may give better results while with another if may be impossible to automate specific controls within the application.

Nevertheless, the most important factor to consider when assessing the ability to automate the TC is the goal of the automation. For this reason, the expert shall keep a clear vision on automation goals at all times.

TEST STEPS

Before getting into step evaluation, it may be worthwhile clarifying the concept of test steps. There are probably other (and maybe better) definitions in the literature, but the definition that follows, which is valid both for manual and automatic testing, should be precise and accurate enough for the method described:

"A test step is an action in a specific context that has a predictable result which has to be verified"

Hence, to achieve correct testing, it shall be accomplished that:

- Context is appropriately set up,
- Action is effectively carried out,
- Expected results are correctly defined so they can be compared
- Verification outcome is unambiguous.

Finally, the combination of several test steps related to functionality constitutes what we call a TC. As commented before, these points also hold true where testing is to be automated. Therefore, the expert should focus on evaluating how each point is accomplished by means of automation in order to assess the ability to automate the whole TC.

TEST STEP EVALUATION

There are several factors for the expert to consider when assessing the automation suitability of a step, which are essentially related to its quantitative and qualitative cost. A schema of those factors can be seen in figure 2.

- DEVELOPMENT TEAM COLLABORATION
- DATA TYPES
  - ➤ Standard vs Custom
  - ➤ Simple vs Complex
- GUI CONTROLS
  - ➤ Standard vs Custom
  - ➤ Simple vs Complex
  - ➤ Command line alternatives
- DATABASE
  - ➤ Standard vs Custom
  - ➤ Direct Access vs Intermediary tool
- AUTOMATION TOOL
  - ➤ Compatibility and language
- OTHER?

The first and most important factor which reduces the cost of step automation is development team collaboration. It should be considered though, that a development team that is ready to assist the testing team in any automation problems is always a good start for automation. On the other hand, uncooperative teams or third-party software leaves the automation solely to the script writer's ability to deal with any problems the application may present.

Another important factor to consider is the data type involved. It must be distinguished between standard data types like strings or txt files and custom data types or custom file formats, which may require extra effort. At the same time, using simple data types like integers or strings shall suppose an easier automation than using complex data types such as images or files.

A third factor to consider is the graphical user interface (GUI). Nowadays, where most applications have GUI, a lot of effort is placed in user friendliness, usability and, in some cases, fancy-looking screens. As a result, in order to assess automation cost, the GUI controls access by automated means becomes decisive.

Usually standard windows controls like buttons or file saving dialogs are easy to deal with. On the contrary, custom-designed controls which may be excellent from the usability point of view can become a nightmare for the test script writer. In addition to this, simple controls such as buttons and edit boxes can be opposed to complex, multifunctional controls like grids. It is not a surprise that third-party custom-designed grids traditionally hold a good position in the top 10 list of test-automation headaches.

Fortunately, for a number of applications designed today it is still possible to skip the graphical interface and use command line alternatives. Command line alternatives may be used as workarounds for custom GUI interfaces, whilst at the same time allowing easy, fast automation. In some cases, GUIs can be tested manually, while underlying functionality may be automatically tested in depth by means of command line interface.

While basically the three above-mentioned factors are the decisive factors in most cases, a few more factors should be taken into consideration. Since a significant number of applications work with associated databases, it must be considered how the application deals with the DB. On one hand, the DB type must be considered. Custom databases may prove hard to test, while a standard DB type may prove easier for most automation tools. On the other hand, DB access may have an impact on automation, since it may require the use of intermediary tools to access it.

Furthermore, the automation tool to be used should also be considered. The behavior of the system, when running both the automation tool and the application under test, shall be assessed for incompatibilities and performance issues. A race for resources must be avoided. Likewise, the complexity of the programming language, combined with the knowledge and experience of the script writing team, are also factors of interest.

Finally, other factors may be evaluated depending on specific projects. For most projects the described factors already provide the expert with enough information to ensure a valid evaluation. Nevertheless, very specific cases may take into consideration additional factors such as criticality, availability of test data, etc…

TEST STEP CATEGORIES

Based on these considerations, the expert evaluates and classifies each step into one of the three following categories:

EASY STEPS: These have a direct, immediate automation script implementation. They usually involve using simple standard data, controls and actions (e.g. verify that a window is closed after clicking a standard "close" button).

COMPLEX STEPS: Those are not difficult to automate by themselves, but require a longer time to code due to their compound nature. A complex step can be understood as a simple manual step which requires several automated

Application Security Testing

4 days testing course by Manu Cohen-Yashar

March 23 - 26, 2009 in Berlin

Limited places

We are living in a world of data and communication, in which the most valuable asset is information. There is no doubt that today's applications must be secure.

Security Standards are created to insure that products will implement security measures to protect their data.

Security is an "all-inclusive" term, which means it must be implemented "everywhere", in all levels:

- **Users**: Train your users and build awareness to help them to reduce the risk of performing irresponsible actions which will be used by the attacker. Make sure your UI helps your user to take the correct decisions.
- **Infrastructure**: Firewalls, Network Admin, Host & Server Hardening, Network traffic encryption etc.
- **Application**: Authentication, Authorization, Input validation, Encryption, Configuration management, Parameters manipulation, Auditing, Error Handling etc.

The application must be designed and implemented while taking security issues into consideration. We have to remember that the attacker needs to find just one security breach while we have to protect everywhere.

Leaving one of the above levels unhandled will result in a completely unsecured product.

Application security is not just another feature. You cannot just turn it on.

Application security demands a lot of thinking. Threat modeling and a lot of design work must be done. Many concrete actions must follow in every phase of the development cycle.

Security Testing:
Testing is a crucial part of Security Development Lifecycle.
The tester must understand methodology of secure development. He has to build a security test plan using the threat modeling documentation.

The tester has to understand the hacking mechanics. He has to get out of the box and think like a hacker.
The tester has to know the security testing methodology. The hacker must be diligent and work systematically to find security breaches.

All this and more will be taught in the course.

2250,- EUR
(plus VAT)

testing experience
Knowledge Transfer

Please register by
email kt@testingexperience.com
or fax +49 30 74 76 28 99

actions/verifications to be fulfilled. Indeed, it could be equivalent to a group of easy steps. They usually involve custom controls that are somehow recognized by the application, relatively complex algorithms or several applications are involved (e.g. step requires verifying contents of several specific fields in a generated txt file).

DIFFICULT STEPS: These prove to be either too difficult to effectively automate or they are nearly impossible to automate. This does not mean they cannot be eventually automated. Instead, the effort needed to automate them may be unacceptable. This is usually due to unrecognized custom controls, actions that cause conflicts, compatibility problems between the target application and the automation tool, or the verification of functionalities which cannot be deterministically verified (e.g. a custom graphical control that plots the result as a graphic and whose numeric results are included as part of the image in almost unpredictable positions).

Part II. Quantitative Classification

After the TC has been evaluated, the expert has a list with all test steps contained in the TC classified either as easy, complex or difficult steps.

In order to provide a quantitative cost of each TC, a unitary cost is defined for each step type according to the previous classification. Thus, unitary cost for easy steps is indicated by the use of c_{EASY} and for complex steps by $c_{COMPLEX}$.

It could be argued that complex steps could be decomposed into a number of easy steps and provide an escalation constant that would multiply c_{EASY} instead of using a second unitary cost. However, for practical reasons, an approximation with an average cost provides a faster method for estimation. This does not forbid a more exact approach by calculating a more faithful estimation by determining the exact number of easy steps implied in each individual complex step if appropriate.

As for difficult steps, it is not easy to provide a reasonable unitary cost. The problem behind each step may be unique and completely unrelated to other steps. Nonetheless, some difficult step may not even be automatable at all in the end. It is suggested to give an expert estimate of automation of each individual difficult step if required. However, we consider it more fruitful to treat difficult steps by qualitative means.

Naturally, the key in quantification relies on the setting of both unitary values. As with many engineering costs, unitary costs are a compromise value that takes into account several factors, which range from the automation tool, the script language and its process, the quality of the script writer team and especially, the historic metrics from previous similar projects.

Once both costs are set, the cost of each TC can be finally calculated. The cost formula for the Total cost for a TC is based in a simple weighted sum, were the total number of easy steps is multiplied by the unitary cost and added to the total number of complex steps multiplied by its corresponding unitary cost as in [1]

[1] $$T_{TC} = \sum n_{EASY} \cdot c_{EASY} + \sum n_{COMPLEX} \cdot c_{COMPLEX}$$

However, it happens often that TCs for similar functionalities within the same functional area are very similar to each other. It can be assumed that in such cases it is quite probable that there is a degree of reusability in the test script. If this is the case, it is suggested to group TCs and single out a representative TC for evaluation. The expert will then define a differentiation constant for each TC in the group, ranging from 0 (no difference) to 1 (completely different). This differentiation factor accounts for the estimation of script variability between TCs and impact on maintenance. TC Group total cost will then be calculated as in [2], where TTC is the cost for the representative TC and kdiff stands for each differentiation factor.

[2] $$T_G = T_{TC} \cdot \left(1 + \sum k_{diff}\right)$$

Finally, a third case is considered regarding cost quantification. In some cases, where the same difficult step is found in several places, it may be worth to overcome it with workarounds. One way to avoid these difficult test steps is by means of auxiliary functions. Auxiliary functions are pieces of script intended to provide an alternate means for accessing a control or executing an action that is considered difficult to automate. Since these auxiliary functions can be equivalent to a TC, a TC cost is calculated for them with step evaluation, unitary costs, etc… The cost of the auxiliary functions is added to the grand total, as if they were another TC. Afterwards, auxiliary function costs are also added to the TCs where they are used modified by a integration cost factor. This factor, ranging from 0 to 0.5, describes the adaptation cost of the auxiliary function, as well as its impact on script and code maintenance. A value of 0 stands for a direct simple call to a simple function, while a value of 0.5 to a more complex scenario. In summary, in cases where auxiliary functions are present in a TC, equation [1] is replaced by equation [3] and the auxiliary function cost is added.

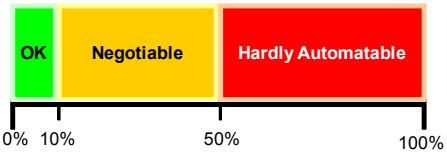
[3] $$T_{TC} = \sum n_{EASY} \cdot c_{EASY} + \sum n_{COMPLEX} \cdot c_{COMPLEX} + \sum \left(c_{AUX} \cdot T_{TC_{AUX}}\right)$$

TC costs can be presented in the form of a table or spreadsheet for stakeholders to decide whether the quantitative costs make the TCs worth of automation or not.

Part III. Qualitative classification

On the contrary, qualitative categorization is based on the ratio of the different step types rather than on their actual number. As it can be deduced from their definition, there is no difference between easy and complex steps from a difficulty point of view. Complex steps can be seen as a group of easy steps. Therefore the key figure to qualitatively determine the automation suitability of a TC is indeed, the ratio of difficult steps among the total steps.

As seen in figure 3, three different intervals are considered. A difficult step ratio below 10 percentage points defines a TC as "OK" for automation, i.e. a good candidate for automation. On the other hand, a difficult step ratio of over 50% indicates a "hardly automatable" TC, i.e. the TC is ill-designed for automation. Execution should be kept manual if possible. Resources should only be spent on script and on keeping the automated TC updated for extremely good reasons.



TCs with a difficult step ratio between 10 and 40%, which are classed as "negotiable", require deeper analysis. It is advisable that the expert comments the pros and cons for each TC on an individual (or TC group) basis. Comments might be critical to tip the scales one way or another. First and foremost, it is of great interest for the stakeholders to know if the automation of the TCs involved will play a key role in achieving the automation goals.

Other factors to consider for comments are:

- whether the use of specific difficult controls is widespread in the application, or whether it is specific to a single TC (e.g. calendar controls),
- whether difficulties within the TC are located mostly in only one of the parts of each step (e.g. result collection or verification),
- whether or not workarounds are available to avoid those difficult steps and test the same functionality (e. g. use of keyboard instead of mouse and vice versa),
- whether the difficult steps could be overcome through the use of auxiliary functions, and
- whether the automation code maintenance costs may be reasonable or not.

If the expert clearly states all factors involved in each 'negotiable' TC, it will result in an easier and more fruitful decision by the stakeholders. In this case, TCs can be presented as a list, table or spreadsheet with the appropriate comments. Obviously, if TCs have been classified both qualitatively and quantitatively, both results can be presented together for a broader view.

Part IV. To automate or not to automate

Once assessment and categorization are completed, a list, table or spreadsheet with all TCs and their associated quantitative and/or qualitative costs is available for the stakeholders to decide whether to automate or not.

The first and most important thing for the stakeholders to take into account before deciding on the automation of all, some or no TCs is the automation goal pursued. Depending on how the goal is expressed, this process may be easy or difficult, fast or slow.

If the quantitative cost of automation is considered, the total automation cost for each TC or group of TCs is reviewed. There are some formulae in the literature to calculate the ROI based on aspects like the average cost of manual execution, and the specification cost [4]. In this case, the goal may be expressed also quantitatively and the decision is immediate.

On the other hand, decision based on qualitative classification may not be that immediate. While candidate TCs with difficult step ratios below 10% or beyond 50% are easily accepted or discarded, those under the negotiable umbrella are to be dealt on a specific basis. Automation goals will filter out a number of them. But afterwards, further arguments may be required for cases still under discussion. The presence of auxiliary functions and their spread within the test suite usually becomes the key factor to decide. If the ratio of difficult steps is not extremely high and the absolute number of steps is reasonably high, partial automation may also be advisable. It may also be considered in cases where only a part of each test step is difficult to automate (e.g. printed reports verification).

Final Notes

The method presented is not bound to any automation tool or SW application. It can be used tailored to any company needs with little effort. We have used it just with spreadsheets, but it can be improved by using a small database. Or it can be made leaner with simple word processor tables. Of course, the unitary cost will need to be defined according to the specific characteristics in each company or software team.

The decision processes of whether to automate or not are accelerated if the method is used systematically. Since the decision is based on historic data, it becomes more solid. Since the method also categorizes test suites, metrics collection is favored and made easier.

[1] Pettichord, Bret “Seven Steps to Test Automation Success“ June 2001
[2] Kelly, Dave “Software Test Automation and the Product Life Cycle“, 2004, Symantec Corporation
[3] Brian Marick, “When Should a Test be Automated?” Proceedings of International Quality Week, May, 1998
[4] Linz, T, Daigl, M. “GUI Testing Made Painless. Implementation and results of the ESSI Project Number 24306”, 1998. Analysis in Case Study: Value of Test Automation Measurement, p. 52+ of Dustin, et. al., Automated Software Testing, Addison-Wesley, 1999.

Biography

A. Farré Benet graduated at the Polytechnic University of Catalonia (UPC) in Barcelona as Electronic Engineer and developed his diploma thesis in Germany, achieving successful results and publishing them in an international physics journal. He also graduated in Computer Science at the Open University of Catalonia (UOC).
After working in research, he joined NTE where he works as senior software quality engineer and test leader in the Clinical Software Division. He presently leads a quality team that tests and validates software for in-vitro analyzers. In the past he also worked as software quality engineer with critical point-of-care applications. His main working fields in software quality assurance are functional testing and system testing of software, risk analysis and FDA audit follow-ups.

Software Testing with T2

by Torsten Zimmermann

How to perform testing jobs using the T2 test framework?
This paper explains the planning of tests and the operational steps with focus on the specification test phases.

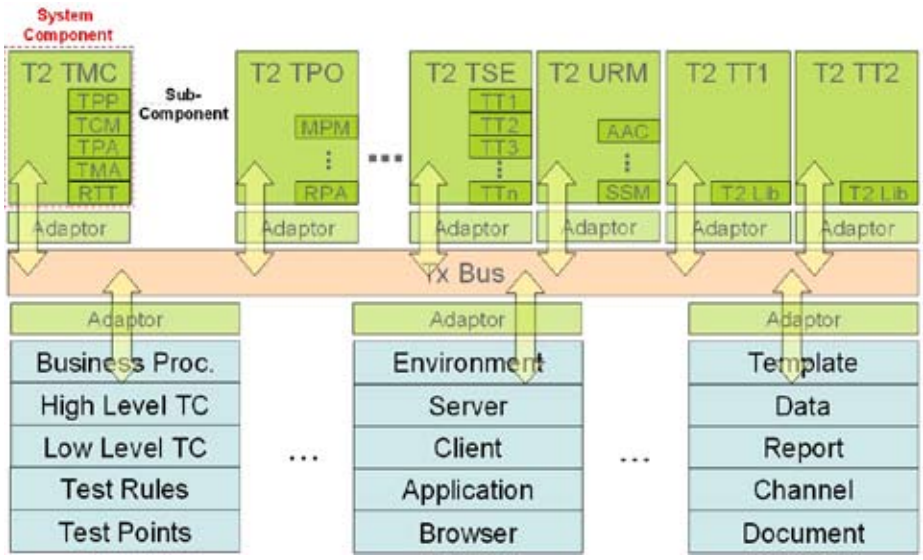
Abstract

The quality assurance of applications is now more than ever a factor that is critical for success. In ensuring the software quality, it is important to follow a comprehensive and structured test concept and approach. This must be an integral part of the design and development process right from the start in order to be able to exclude errors and performance problems from applications even before they go into operation – not an easy task for responsible IT departments. After all, the development cycles for applications are becoming shorter and shorter, the legal framework conditions increasingly complex and the product quality demands higher. Therefore, Torsten Zimmermann invented the T2 Test Framework in the years 1999 up to 2004 to use test and system tools in a very integrated manner. This test tool platform supports the control and execution of extensive tests like integration test, system test, system integration test and system operation tests.
However, in addition to testing and quality assurance requirements, today's state-of-the-art test frameworks must take cost and effectiveness into account to support professional business process requirements. Therefore, the T2 test framework offers features such as test script modularisation and test case orchestration to support cost-oriented testing as explained in this article.

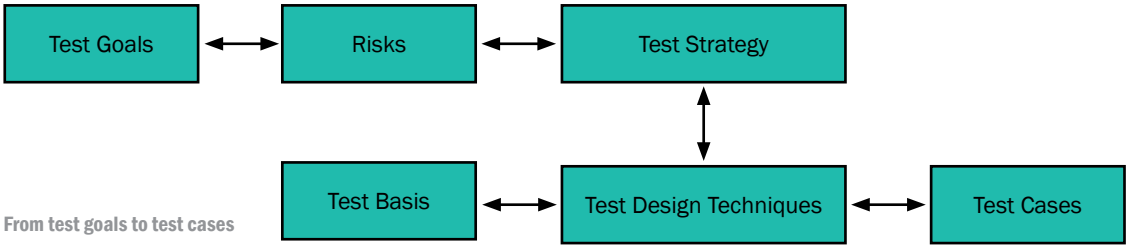
T2 at a Glance

T2 wants to achieve a higher integration of several tools (testing tools, tools for system administration, application integration, coding, etc.) to support all test phases of the fundamental test process (planning, control, specification, preparation, execution, evaluation and completion). A higher integration of processes and tools within one framework over the entire test process scope leads to increased benefits.
Thus, the test framework offers better customizing capabilities, better integration of test tools from different suppliers, and valid test environments. No proprietary script languages for automated and manual tests are needed, because all test scripts / test procedures and steps can be designed with the T2 Test Modeling Language (T2 TML). This cutting-edge solution represents a significant reduction of know-how transfer efforts. Test engineers pass a 4-hour T2 TML introduction and tutorial package followed by 4 days of training on the job: after one week of training and coaching, the test engineers are able to define the test steps in T2 TML.
Today, the T2 architecture is based on a mixed service bus - the Tx Bus - and represents a very flexible option to easily integrate further tools and systems to support the complete testing operations.

As shown in the chart, the Tx Bus communicates with every T2 system component (daemon concept). Every system component communicates with other system components via the Tx Bus. The tool domain manager within T2 Test Environment Management (T2 TEM) administers all available tools with their active / inactive states. Therefore, the Tx-Bus is able to reject messages with unknown or inactive addressee.
The generic test script approach of the T2 TML is one of the most important features of this language. It allows test case design in a tool-independent modeling language. This way,



T2 TTP architecture principles



From test goals to test cases

test cases do not have to be designed for every available test driver (test automation tool). Besides this, the same T2 TML script can be executed for manual and automated testing without any script modifications. Several test tools – also of different test disciplines – can be used within T2 TTP by using the T2 Test Modeling Language (TML) to define all tests (functional tests, load tests, performance tests, interface tests etc.).

Therefore, the T2 TML command reference is structured in several language modules. The T2 TML offers a flexible structure by using the following key values:

- **Core language**: The core language represents the command core, which can be used in every test discipline and platform technology. Language functionality like logical expressions and commands (e.g. the if-then-else construct) or variables respectively value assignments to variables are typical parts of the core language.
- **Language modules**: Every supported test discipline needs at least one additional language part (language module) which reflects the test discipline related command reference. E.g. load tests need special commands which functional tests would not need. The available modules are:
  1. GUI: This module represents the functional testing via graphical user interfaces for the system and system integration test levels.
  2. EMAIL: This module represents the communication testing. Right now, only the communication with email servers is established.
  3. SOAP: This module supports the interface testing of web services at the function integration, system, and system integration test levels.
  4. DB: The DB module enables to interact directly with the DBMS to support the direct CRUD tests within the respective database.
- **Customizable language grammar**: The language requirements mentioned in the preceding paragraphs lead to the customization point. It is helpful to customize the T2 TML language grammar to adapt the T2 TML to currently existing test lab conditions. This way, the modeling language will be able to fit the test requirements more closely. The language grammar is stored in XSD files. Thus, the T2 TML customization can be performed during migration procedures without having to rewrite code for any T2 TSE components.

The Big Picture of Test Specification

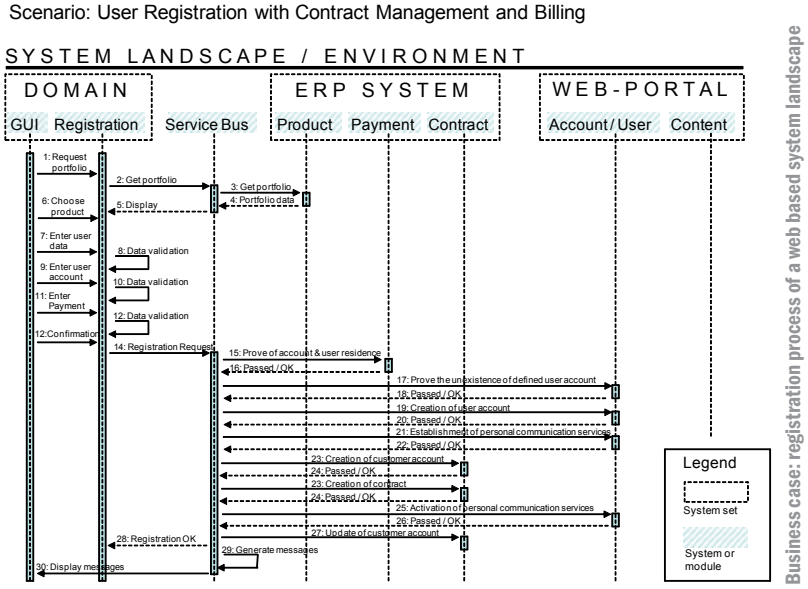
In this chapter, I would like to explain how to specify and prepare tests using common test design techniques. Due to its flexibility, the T2 TTP supports several design techniques by integrating the respective test case design tools into the test tool platform.

The test cases are derived from the test bases using test design techniques whereas the test strategy defines what kind of design techniques should be used. The test strategy is chosen in accordance with the available risks. The test goals determine the importance of every risk.
A test goal defines the success criterion for the test project. Product risks depict the chance the product fails evaluating the damages and impacts to the business process in case of occurrence. The test strategy defines the way to find the most important defects as early as possible at the lowest costs. The test intensity is light, moderate, or thorough. The intensity should be chosen in accordance to the product's maturity. For an early stage product may be a light coverage sufficient whereas the thorough coverage is needed for high sophisticated products, which are out on the market in productive environments for several years.
There are a lot of design techniques available. The following table shows an overview of the most used techniques:

Characteristic	Test intensity		
	Light coverage	Medium coverage	Thorough coverage
Manageability	PCT / test depth level 1 UCT / checklist EG	PCT / test depth level 2 DCoT / equival. classes ET	PCT / test depth level 3 DCoT / pairw. testing
Security	EG	DCoT / equival. classes SEM / MCDC ET	DCoT / pairw. testing PT
Usability	UCT / checklist EG	UCT / paths PCT / test depth level 2	UCT / decision points RLT / operational
Continuity	EG	RLT / operational ET	RLT / operational RLT / load profiles
Functionality (overall)	DCoT / equival. classes UCT / checklist SYN/ checklist	DCoT / pairw. testing DCyT / CRUD DCyT / decision coverage	DCoT / N-wise testing DCyT / CRUD, extra RS DCyT / MCDC
Functionality (detail)	DCoT / equival. classes DTT / DCDC ECT / DCDC EG	DCoT / pairw. testing ECT / MCDC ET	DCoT / N-wise testing DTT / MCC ECT / MCC
Functionality (validation)	SYN/ checklist EG	SYN / prioritised list SEM / DCDC	SEM / MCDC
Infrastructure Performance Efficiency	EG	RLT / operational RLT / load profiles ET	RLT / operational RLT / load profiles
User-friendliness	SYN/ checklist EG	PCT / test depth level 2 SYN / prioritised list UCT / checklist	UT
Suitability	PCT / test depth level 1 UCT / checklist DCoT / equival. classes EG	PCT / test depth level 2 UCT / paths DCyT / CRUD DCyT / decision coverage DCoT / pairw. testing ET	PCT / test depth level 3 UCT / decision points DCoT / N-wise testing DCyT / CRUD, extra RS DCyT / MCDC RLT / operational RLT / load profiles
Portability	Checklist Random sample FTs Random sample environment combinations EG	Functional regression test Important environment combinations ET	All FTs All environment combinations

Proposed technique for a specific combination of characteristics and test intensity (with courtesy of Leo van der Aalst)

The table above shows the most common test design techniques, which are explained in [Aal08]. The test example deals with the user registration of web-based application respectively system landscape. The respective business process is shown in the following sequence diagram:



The analysis of respective requirement / design specifications, user / operation manuals, online help information etc. represents the foundation of good software testing. To get a closer picture of the system landscape and the respective business processes, they need to be tested. Further interviews with product and solution managers as well as development-oriented teams help to prove the conclusions made during the analysis phase. At least, this data leads to several sequence diagrams with further testing-oriented documentation. Based on the named chart, one important result is the determination the business process:

- Selection of product
- Enter user data
- Enter user name and password
- Select payment and enter payment data
- Data confirmation
- Creation of user account
- Creation of contract
- Generation of messages

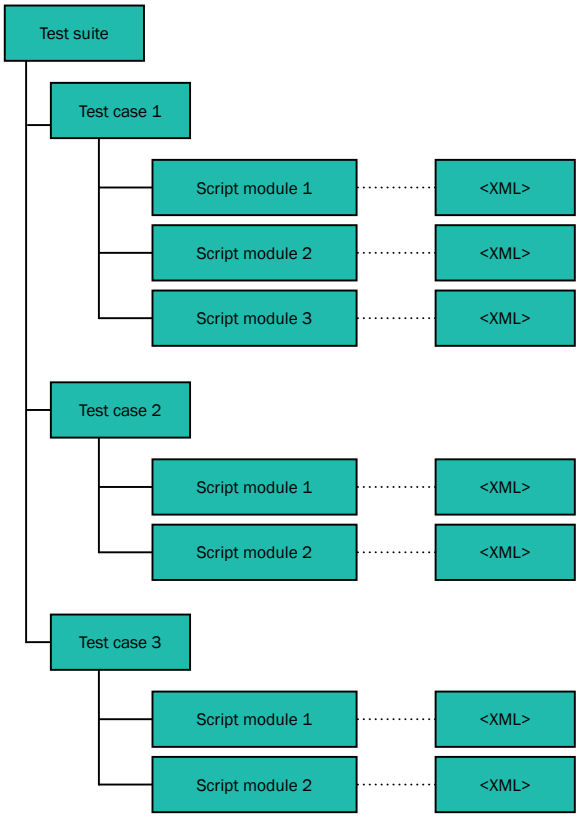
This list already defines the script modules, which need to be elaborated and assigned with the respective test cases in the TMC (Test Management & Control). By the way: The [Associated test script modules to respective test cases] chart in the Test Case Orchestration chapter shows the composed test case content in the TMC (Test Management & Control) which complies to this analysis result. The test management and control phase represents initial and accompanying activities. Its goals are the planning and control of needed tasks, resources and tools to execute the defined test procedures in the correct manner. Furthermore, the planning phase is vital to define the actual testing strategies, goals and design techniques. These points are documented in the related test plan. For example, the usage of pair-wise test design technique in combination with comparison tests may be a common approach for detailed or overall functional testing. Of course, it represents a usual proceeding in systems integration testing. For better understanding, the following chart shows the relationships between the test suites, test cases, the script modules and the T2 TML scripts.

The T2 test suite contains the test cases as identified during the test case design activities. This part is usual. The test cases themselves are defined by test script modules, which represent a (business) process or user activities. The module scope depends on the test scope and goal. A correctly adjusted module scope supports a high reusability rate of the established modules. Therefore the same module can be used in several test cases as the graph shows. Every module represents a container which is linked to a T2 TML script. The respective TML script contains the T2 commands and resides at the TMC server. The test manager can compose the test cases by defining the respective script module sequences without any coding. This procedure can be done by drag-and-drop of the selected module from the module library to the test case tree within the TMC (Test Management & Control) system. It is called test case orchestration which is explained later on in this article.

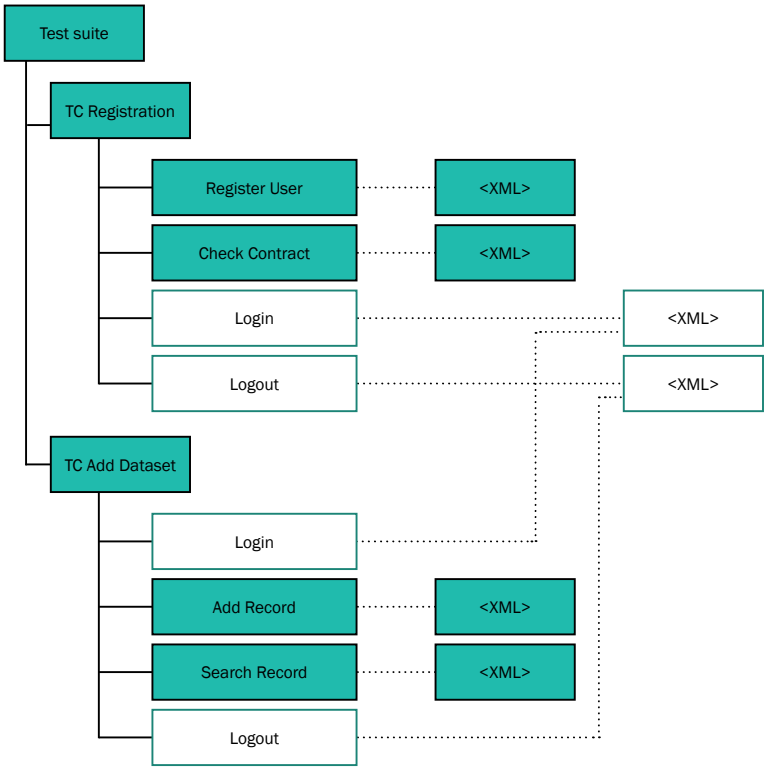
Test Case Orchestration

Using CaseMaker, the creation of test projects, test suites, and test cases will be less important for the test manager due to the reduced manual tasks. (I would not like to say that this part might be unimportant). Besides the higher reusability of the designed test script modules compared to holistic test scripts, the test case orchestration offers the opportunity to perform script design and test management tasks in parallel. Of course, the test cases depend on the script modules and without available scripts no test runs can be executed, but the scripts are not needed during the composition process. The test cases are solely connected to the script containers. Therefore, the script modules can be assigned to the respective script containers at any time.

Principle structure of T2 test suites



Example for script module reusability: the modules login and logout can be reused.



What about your intellect?

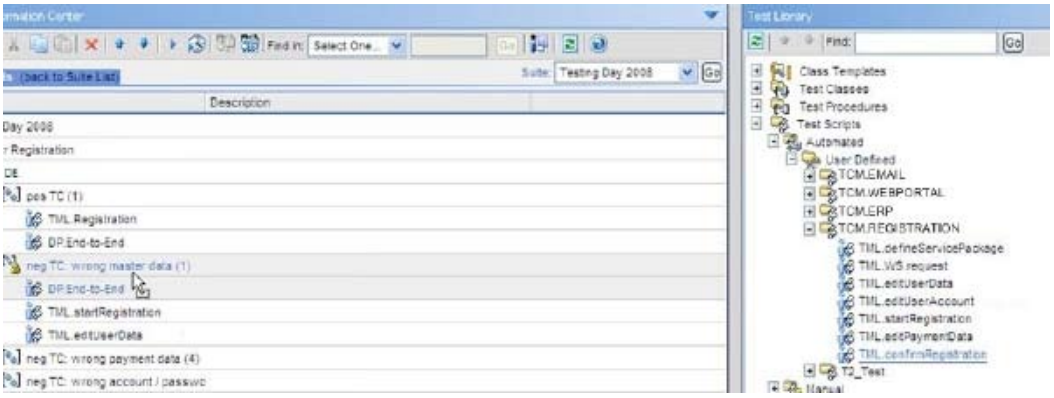
CaseMaker – the tool for test case design and test data generation

If you are ISTQB Certified Tester, you can get a **free version** of the **new** client software for **12 months**.
Please register at www.casemaker.eu/istqbct

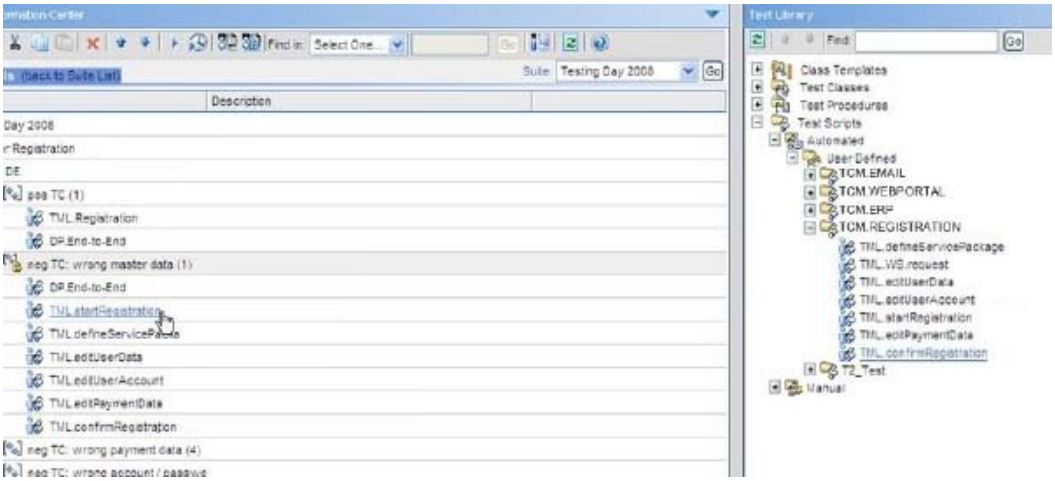
ISTQB
Certified Tester?
Get a FREE
Version

CaseMaker®

www.casemaker.eu

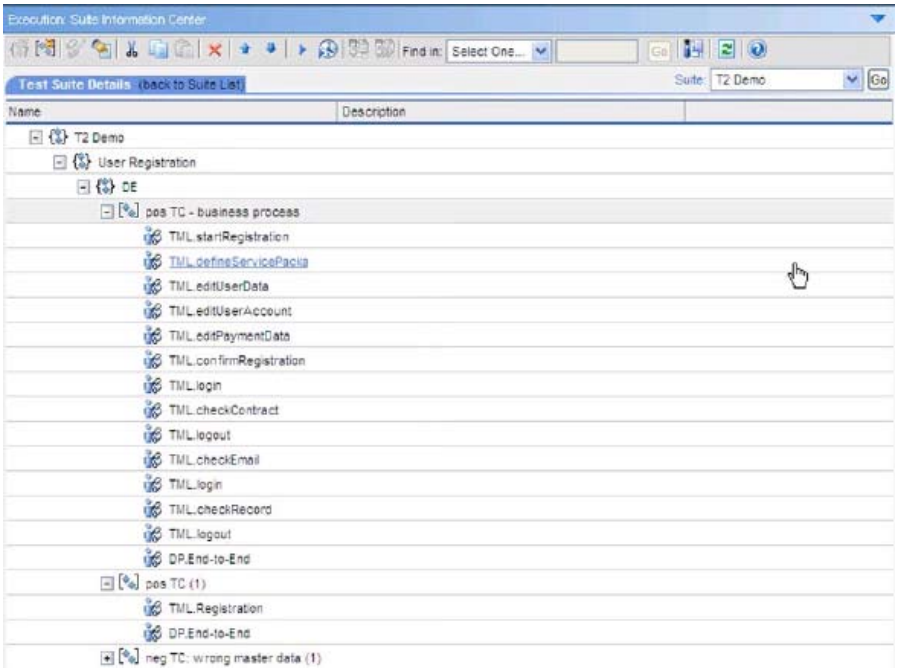


The test case (left) can be composed step by step using the script...



...modules (user actions or business process steps) available in the test library (right).

Test managers and coordinators can now compile their needed scripts for respective test cases together without a need for any programming skills. The following hardcopy shows a finalized test case composition for the positive case of user registration testing:



Associated test script modules to respective test cases (test case orchestration) in a test suite

As you can see, the test case "pos TC – business process" contains several test script modules, which clearly describe the test process and its steps accompanied by the data pool (dp.end-to-end) containing the primary test data / test parameters. The test case "pos TC (1)" below represents the same test case, but designed in the old-fashioned holistic manner.

T2 TML Code Example

The T2 TML test script defines the test steps, which can be used for one or several test cases. The usage of data pools (primary test parameters) ensures that the test script can be engaged for more than one test case due to its higher adaptation and control capabilities. It enables the usage of explicit (e.g. < *gui: startApplication =”IE” />*) as well as relative assignments (e.g. *<gui: click object = “$env.WebPortal.loginbutton”” />*). This example uses data pools and object repositories within its test object definitions. The relative assignments are recommended to support abstraction layers at the full scale. This means that the test script can be designed in parallel to the respective software development. Object repositories and data pools (primary test data, test parameter) can be added after the test script design, when test object and secondary test data are available.

Within T2, the usage of script modules is preferred instead of holistic scripts, since it reduces elaboration and maintenance efforts. This means that test engineers and specialists for test automation elaborate solely components, which have been put together in the test orchestration by test managers or coordinators to setup the test suites and complete its test cases.

```
<?xml version="1.0" encoding="UTF-8"?>
<testcase xmlns="http://url.com/xsd/tml/testcase"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:core="http://url.com/xsd/tml/module/core"
        xmlns:gui="http://url.com/xsd/tml/module/gui"
        xsi:schemaLocation="http://url.com/xsd/tml/testcase ../../T2/common/xsd/tml/testscript.xsd
                            http://url.com/xsd/tml/module/core ../../T2/common/xsd/tml/modules/core.xsd
                            http://url.com/xsd/tml/module/gui ../../T2/common/xsd/tml/modules/gui.xsd"
        id="LOGIN"
        title="Login - with datapools"
        isNegativeCase="false"
        status="beta"
        usesDatapools="true">

    <description>
        <short lang="en">Login Module</short>
        <intention lang="en">This script is part of an end-to-end presentation</intention>
        <detailed lang="en"/>
        <history>
            <entry date="12.06.2008"><editor>Torsten Zimmermann</editor><log>initial version</log></entry>
        </history>
    </description>

    <variables>
         <variable name="tryresult" type="boolean"/>
    </variables>

    <conditions/>

    <gui:start application="IE"/>
    <gui:maximizeWindow/>
    <gui:setURL value="$env.WebPortal"/>

    <core:try result="$tryresult">
          <gui:click object="$env.sec.link"/>
    </core:try>

    <gui:use objectgroup="env.WebPortal.usernamepassword" do="set" with="dp.usernamepassword"/>
    <gui:click object="$env.WebPortal.loginbutton"/>

    <core:try result="$tryresult">
          <gui:click object="$env.WebPortal.firstlogin"/>
    </core:try>
</testcase>
```

Legend

XML Header

Test case wrapper

Test case description and change log, IEEE 829 compliant

Declarations and conditions section, IEEE 829 compliant

Test command section, Definition of test steps

Example of aT2 TML script module

This TML script example above defines the login activities (→ TML.login). The red text marks the TML commands, which define the respective test steps and activities. The TML based script modules can be transferred to almost every test driver available on the market. Using TML means to run the defined test step definitions in the automated or manual execution manner without the need to have in depth test tool and test driver know-how: Just use T2!

Appendix

Glossary

DCoT	Data Combination Test
DCyT	Data Cycle Test
DTT	Decision Table Test
EG	Error Guessing
ET	Exploratory Test
ECT	Elementary Comparison Test
FT	Functional Test
FIT	Functional Integration Test
PCT	Process Cycle Test
PT	Penetration Test
RLT	Real Life Test
SEM	Semantic Test
SYN	Syntactic Test
UT	Usability Test
UCT	Use Case Test
CRUD	Create, Read, Update, Delete
DCDC	Discrete Condition / Decision Coverage
MCC	Multiple condition coverage
MCDC	Modified Condition / Decision Coverage
TEM	Test Environment Management
TMC	Test Management & Control
TML	Test Modeling Language
TSE	Test Script Engine
TTP	Test Tool Platform
Tx Bus	Service Bus of T2
SDS	System Design Specification
SRS	System Requirement Specification
SUT	System Under Test
TC	Test Case

T2 TML Command Reference

CORE

if – then – else	exit	logPassed	logFailed
delay	log	run	try
skip	debug	store	restore
check	parse		

GUI

check	checkRegExp	checkProperty	checkExistence
set	get	click	focus
getURL	setURL	clearText	waitForChange
waitForAppearence	focusWindow	start	getWindowID
getWindowName	killAllApps	closeApps	killApp
maximizeWindow	minimizeWindow	use	test

SOAP

connect	send

EMAIL

connect	send	to	cc
bcc	content	check	checkAndDelete
waitForMail			

References

[Aal08] L. van der Alst: Test design techniques were not invented to bully testers!, Testing Experience, September 2008

[Bal98] H. Balzert: Lehrbuch der Software-Technik, Spektrum Akademischer Verlag, 1998 (especially Chapter 5 and 6 of the book Software-Management, Software-Qualitätssicherung, Unternehmensmodellierung)

[IBM05] Professional test automation with IBM Rational Robot, IBM, 2005

[Pol98] T. Pol, T. Koomen, A. Spillner: Management und Optimierung des Testprozesses, dpunkt Verlag, 1998

[Pol02] T. Pol, R. Teunissen, E. van Veenendaal, Software Testing, Addison-Wesley, 2002

[Sim06] F. Simon, O. Seng, T. Mohaupt, Code-Quality-Management, dpunkt Verlag, 2006

[Zim01] T. Zimmermann: Testen will gelernt sein. QZ Magazin, Hanser Verlag, 2001

[Zim05] T. Zimmermann: Testen kostet Geld – nicht zu testen auch! Business-Wissen.de, 2005

[Zim07] T. Zimmermann: The T2 Effect, Business Process Engineering, d-punkt Verlag, 2007

Biography

Torsten Zimmermann startet his professional career in 1993 after completing his degree as ‚Diplom Wirtschaftsinformatiker'. Since 1995 he has been contributing to international projects in the area of software quality and test management at various corporations, amongst them BMW, DaimlerChrysler, Hewlett-Packard and Hoffmann-La Roche.

He has developed the risk-based testing approach which has been amongst others presented in the professional publication 'QZ' and today is established as state-of-the-art knowledge in the area of software quality assurance. His accumulated experiences have led to the creation of the T1 TFT (Test Framework Technologies) which stands for the advent of a new area of testing systems.

Today Torsten Zimmermann is developing new approaches to enhanced test concepts and test frameworks. In cooperation with a network of universities this creates new solutions for rules- and model based testing systems.

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

160.000 readers

ADVERTISEMENTS
sales@testingexperience.com

SUBSCRIBE
www.testingexperience.com/subscribe.php

PRICE
free of charge

ISSN 1866-5705

Index Of Advertisers

Yaron's Forecourt
Test Automation – What's going on behind the scene?

by Yaron Tsubery

Dear colleagues and readers,
Welcome back to our joint forecourt, I hope that you're already equipped yourselves again with a glass of chilled wine or a pint of cold beer… I've ordered the snacks already. I hope that you're in a good mood, willing to read, and to share your thoughts on today's subject: Test Automation …
I first came into contact with this after I had decided to become a developer many years ago – you know the sort of thing: write code, create a program, and - if you're well trained then you've learnt to write some automatic code that will test on its own. Sitting in the garden one day with my colleagues discussing the subject of automation, you can imagine that there were many different opinions and many things to take into account (because some of us came from different domains within the company). Mature organizations do, of course, use automation; those that test embedded systems can achieve more test automation coverage; those that deal with mission-critical systems better had test automation running from day one. So, you can see that domains can have an impact on the need for automation, too – sometimes more than what is already known as the major impact on investing in it or not – MONEY.
While we talked in my forecourt here, you could also witness different perspectives in relation to test automation (bearing in mind that we do come from different domains in the company), such as: What does your manager think about it? How do you convince him to invest in test automation? Of course, there are freeware automation tools available in the market; but not all of them are able to support our needs in automation, and even if they do, we still have to invest in development time in addition to the usual manual tests… . Well, most of us know that we have to do some research and present details about test coverage, costs and show the ROI (Return Of Investment). So, a colleague of mine got up from his chair very quietly and said: "…you know, it's not enough to convince your manager. In big organizations there are regulations and standards which must be followed, before you can get your manager's approval…." He then gave us some examples of the standards for selecting a tool in their company. Then a lady from the development department raised her small voice and asked: "Who is responsible for programming the test automation in your company?" Silence… followed by a very loud discussion. You can imagine why… developers thinks, it's in their domain and we, the testers,think it's in ours; after a while one of the guys who led several test automation projects stopped us and said: "…R&D managers, who decided to have test automation programmed by developers, have after a while come to the conclusion more than once, that it's not for developers to deal with. They saw that it needs test engineers with programming abilities and skills – such as we have in performance tests, the load test engineers – and that they should lead it, "…it's not in our domain…", they said…".
Then we continued to discuss on how a test automation project can leverage the scope of the testing department, the growth of specialized skills that test engineers will need to have. Whilst digging further into the subject, we got a few technical tips too, like, how to implement KDT (keyword driven testing) that can be used in automation – enabling the test engineer to do most of the scripting job (yes, you must first have a test automation tool with a robust framework/platform that will support it). Many other questions were raised at the table, such as: "…did anyone perform a research on the efficiency of integrating a few test automation tools to support the test coverage you want to achieve for the product under test?"; "…what can be conceived as a good average percentage of test automation coverage in a company?"
Many technical questions were raised too, like: "…how to implement it in an efficient way that will cause less maintenance work later on?"; "…What are the best ways to recover from an automated test that failed? (Humans will see that and react accordingly; but we need to "show" the automation tool how we as human react too)… We ended up at midnight, a little bit dizzy but very happy, with lots of thoughts and open questions still unanswered. So, if you have a question or something that you wish to share, don't be shy – send them to the following account, that has been set up especially for our discussions: yaron@testingexperience.com
Yours truly,
Yaron

Biography

Yaron Tsubery has been a Director of QA & Testing Manager at Comverse for seven years. Yaron is in charge of a large group that includes Testing Team Leaders and Test Engineers. The group deals with Functional, Non-functional, Load & Performance tests, some of which are done off-shore. The group has much experience in managing and controlling acceptance tests. Yaron is also known as a manager who has built testing groups in his past and also acted as a consultant to testing managers while they were building their testing groups.
Yaron has been working in software since 1990, and has more then 15 years in testing field as a Test Engineer, Testing TL, and Testing Manager. His original profession was Systems Analyst.
Yaron worked in Product Management, Project Management and Development for at least 3 years before becoming a Director of QA & Testing.
Yaron is a member of ISTQB (International Testing Qualifications Board – www.istqb.org) and is the President and founder of the ITCB (Israeli Testing Certification Board – www.itcb.org.il). He is a member of IQAMF (Israeli QA Managers Forum) and in SIGiST Israel.
Yaron has been invited as a speaker to some important international conferences to lecture on subjects related to testing.
Yaron has lots of experience in banking business processes. For the last 7 years he has implemented best practices in a field where he is in charge of producing complex systems for telecommunication companies such as Vodafone, T-Mobile, NTT DoCoMo, Verizon Wireless etc';
Main systems in Mobile technology: i-mode, Mobile GW and Billing Proxy, in SI projects: Content Data Management solutions, Video Solutions, IPTV and IMS.

3 day testing course by Alon Linetzki

February 25-27, 2009 in Zürich, Switzerland

Increasing Added Value & ROI in Testing

We are facing today an increase demand from test managers to improve productivity and product quality, to reduce costs of testing, to reduce resources, and to make-things-right-the-first-time.
This challenge is forcing test managers to improve in 3 major topics:

- Improve testing processes
- Manage testing project risks better
- Manage professional testers, and the supporting environment better by improving communication, diplomacy and negotiation skills

The course is enforcing those and aiming to educate test managers in how to be better in these areas of concern. The course includes practical exercises and simulations in the relevant topic areas.

The learning objectives of Increasing Added Value and ROI in Testing course focus on the below main areas of practical doing:

- To be able to know what test process improvement and TPI model are
- Discuss a method for evaluating process maturity from different aspects
- Discuss the different dependencies between the testing processes
- Know how to quick-start TPI effort in a testing organization
- Discuss which projects to pick for the TPI pilot (with best chances to succeed)
- Understand the implications of managing risks in testing projects
- Understand the concepts of Risk Management
- Describe Risk Based Testing principals
- Understand what is the Risk language
- Define where RBT can assist during the testing life cycle
- To learn how to exercise an **excel tool** for risk strategy (planning phase)
- Discuss test execution strategy issues and RBT
- To learn how to exercise an **excel tool** for risk analysis (scheduling and execution phases)
- To learn about good communication concept and methods
- To know different paradigms of personalities, and reflect those on us
- Understand how to convince others without getting resistance
- Discuss how to present and pass a message to others
- Discuss how to give and obtain feedback, and get a positive impact

Please register by email kt@testingexperience.com
or fax +49 30 74 76 28 99

Limited places

1450,- EUR
(plus VAT)

te testing experience
Knowledge Transfer

Díaz Hilterscheid

Training with a View

Kurfürstendamm, Berlin © Katrin Schülke

19.01.09-21.01.09	Certified Tester Foundation Level	German	Hamburg
26.01.09-28.01.09	IREB - Foundation Level	German	Berlin
26.01.09-29.01.09	Certified Tester Advanced Level - TESTMANAGER	German	Wien
02.02.09-04.02.09	Certified Tester Foundation Level	German	Berlin
09.02.09-12.02.09	Advanced Functional Tester	German	Berlin
16.02.09-18.02.09	Certified Tester Foundation Level	German	Stuttgart
23.02.09-25.02.09	Certified Tester Foundation Level	German	Berlin
02.03.09-05.03.09	Certified Tester Advanced Level - TESTMANAGER	German	Berlin
09.03.09-11.03.09	Certified Tester Foundation Level	German	Düsseldorf
09.03.09-11.03.09	**ITIL v3 Foundation with exam in cooperation with TÜV Rheinland Secure iT GmbH - NEW -**	German	Berlin
16.03.09-18.03.09	Certified Tester Foundation Level	German	Berlin
30.03.09-01.04.09	IREB - Foundation Level	German	Berlin
06.04.09-09.04.09	Certified Tester Advanced Level - TESTMANAGER	German	Düsseldorf
20.04.09-22.04.09	Certified Tester Foundation Level	German	Berlin
27.04.09-29.04.09	Certified Tester Foundation Level	German	Frankfurt

"Thanks for the entertaining introduction to a complex topic and the thorough preparation for the certification. Who would have thought that ravens and cockroaches can be so important in software testing"
Gerlinde Suling, Siemens AG

"Mr. Lieblang's great wealth of experience, his excellent lecture style and his refreshing manner made the 3-day ISTQB training into an experience. With this preparation passing the subsequent examination was child's play - which our course result of 94.5% clearly proves."
Stefanie Schlegel, T-Systems Multimedia Solutions GmbH

"A casual lecture style by Mr. Lieblang, and dry, incisive comments in-between. My attention was correspondingly high. With this preparation the exam was easy."
Mirko Gossler, T-Systems Multimedia Solutions GmbH

"Thank you for 3 very informative days. I passed my exam with 39 out of 40 questions correct and came away with new inspiration for my work."
Rita Kohl, PC-Ware Information Technologies AG

also onsite training worldwide in German, English, Spanish at
http://training.diazhilterscheid.com/ training@diazhilterscheid.com