16

December 2011

testing experience

The Magazine for Professional Testers

printed in Germany
print version 8,00 €
free digital version
www.testingexperience.com
ISSN 1866-5705

The Future of Testing

Certified Agile Tester

Pragmatic, Soft Skills Focused, Industry Supported

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

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

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

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

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

Book your training with Díaz & Hilterscheid!

Open seminars:

05.12.11–09.12.11 in Berlin, Germany *
13.02.12–17.02.12 in Munich, Germany *
23.04.12–27.04.12 in Berlin, Germany *

30.01.12–03.02.12 in San José, USA
05.03.12–09.03.12 in Helsinki, Finland
19.03.12–23.03.12 in Amsterdam, The Netherlands
11.06.12–15.06.12 in Stockholm, Sweden

* (German tutor and German exam)

Díaz Hilterscheid

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

Editorial

Dear all,

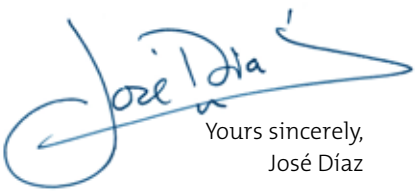
it was a crazy month! In November there was plenty of conference work and work for Testing Experience. The result of the latter is now in your hands. I hope you like the articles.
We are currently preparing a new website for Testing Experience that we hope will fulfill all your wishes ;-)

In a few weeks we will start into the fourth year of Testing Experience. It was great to see how the child grew up. There have been some good and some very good issues of the magazine, and we did always have the support of the community worldwide.
In the meantime Testing Experience is read in the whole world; there is no country with testers that does not appear in the list!
This year, the website had more than 7 million hits! The most frequent download is the September 2009 issue with over 450,000 downloads. Impressive numbers. Please contribute to this success by sending us good articles and spreading the word. Testing Experience is the de facto magazine for testing professionals worldwide.

We recently ran the Agile Testing Days in Potsdam near Berlin. It was great. If you want to read a resumé, please go to http://www.agile-ux.com/2011/11/19/agile-testing-days-2011-day-1-what-a-fabulous-day/. It was really amazing. I would like to again congratulate Gojko Adzic for winning the award "Most Influential Agile Testing Professional Person". The award was handed over by the Lord Mayor of Potsdam. We celebrated the social event and the award ceremony at the Prinz Eisenherz Restaurant at the Babelsberg Studios. Lisa Crispin on a horse did the speech of honor for Gojko. It was great. We had fire games, magic close-ups, an acrobatic show and medieval music. If you were not there, you missed a good conference and a great social event. Never mind, there's always next year...

We are planning the Belgium Testing Days in Brussels (www.belgiumtestingdays.com) and the Testing & Finance conference in London (www.testingfinance.com). Don't miss them if you want to experience a new type of conference. We love knowledge, networking and social interaction. We bring the best to you. Enjoy it!

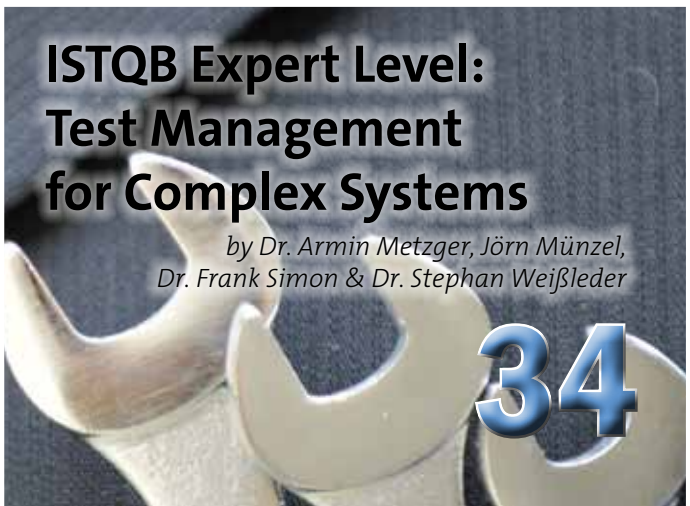
I wish you, your family, friends and colleagues a Merry Christmas and a Happy New Year – peace, health, love and time for you and your family!



Yours sincerely,
José Díaz

testing experience

Contents



Lean Test Management – The future of testing?

by Iris Pinkster-O'Riordain & Bob van de Burgt

1. Lean Test Management: an introduction

Lean is hot. Lean is everywhere. When you are walking down the street, you will find a lot of companies advertising how Lean they are: Lean building companies or Lean health care. You can buy Lean apps and you can hire consultants specialized in Lean. It looks like Lean methods are applied to various types of businesses and hence various types of processes. What they all have in common is the focus on a complete business process and not just improving one of the process steps. For example, the focus is not just on improving the software testing process, but on improving the complete process from requirements gathering until implementing the solution into production. In this article the authors will investigate whether the future of software testing can also be found in Lean. And as Lean is very often referred to in combination with Six Sigma, this method will also be incorporated in the investigation.

Let's first have a look at the principles of these two methods.

2. Lean: to create a better workflow and eliminating redundant activities

Lean manufacturing is a company process improvement method. It is developed from a logistics point of view. Lean manufacturing concentrates on banning waste to become quicker and more efficient. Every process that does not add value for the customer is eliminated. The Lean manufacturing philosophy is derived mainly from the Toyota Production System (TPS). It is renowned for its focus on reduction of the Toyota ‚seven wastes' in order to improve customer value.

The basis of Lean is the **7 wastes** that have to be eliminated. Many of those can be related to testing as well. They are:

- *Overproduction*. Simply put, overproduction is to manufacture an item before it is actually required. Within testing this might be what we want: before the start of the actual test execution the test cases, test data and test environment are prepared. On the other hand are we sure that we make a distinction between effort and depth of testing based on the risks? Do we not have too much overlap between test levels? And what about the potential overkill of test plans? These can all be seen as overproduction within testing.

- *Waiting*. Whenever goods are not moving or being processed, the waste of waiting occurs. Linking processes together so that one feeds directly into the next can dramatically reduce waiting. In testing this is a very common and well known waste: waiting for designs, waiting for the right environment, waiting for support, etc.

- *Transporting*. Transporting a product between processes is a cost incursion which adds no value to the product. For testing we can look at the way information gets to the tester or the amount of meetings there are with the test team.

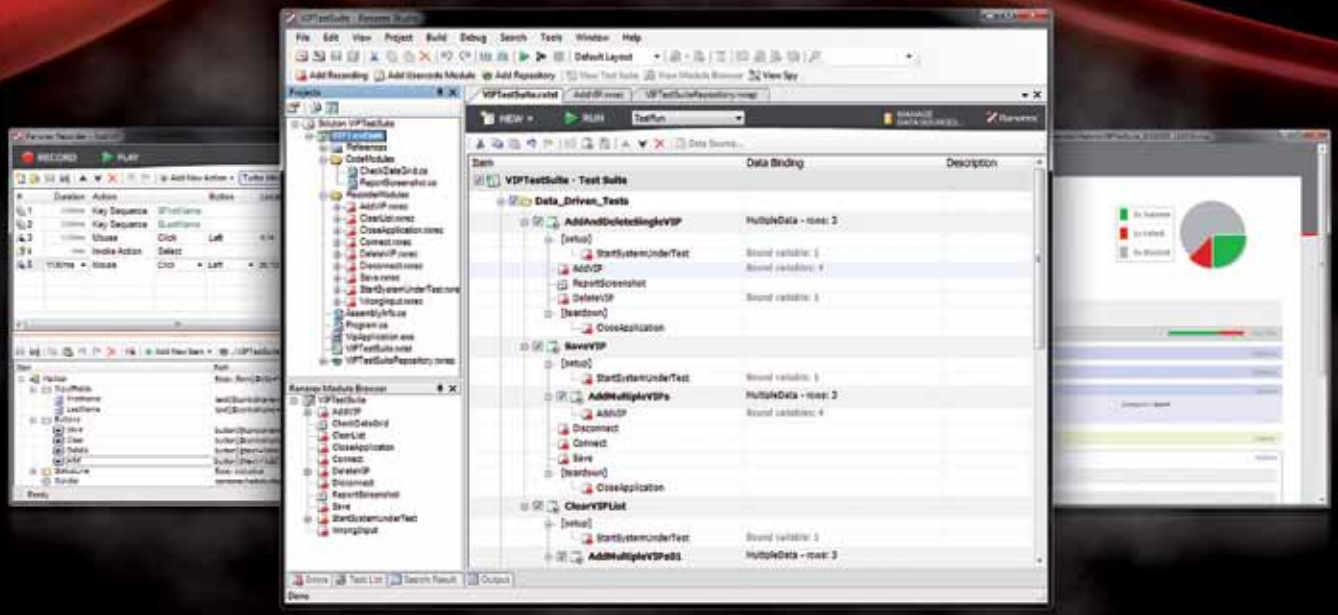
- *Inappropriate processing*. Often termed as "using a sledgehammer to crack a nut", many organizations use expensive high precision equipment where simpler tools would be sufficient. We all might have examples of test automation tools or an expensive defect administration of which a big part of the functionality is not used. And we can even think of having too many test cases prepared for a small and simple functionality when we did not consider the product risks and their priority.

- *Unnecessary inventory*. Work in Progress (WIP) is a direct result of *overproduction* and *waiting*. Instead of having good test cases ending up as shelf ware, good version control can make it possible to re-use test cases. Another example might be to

Ranorex®

Automate your User Interface Testing

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



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

Record and Edit Reliable Test Actions

Manage and Execute Your Automated Tests

Reproduce Bugs and Maintain Your Tests

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

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

look at the test teams and make them more flexible to work within various projects. This helps having the right number of people at the right moment in your projects. Another example is building more test cases than can be executed in the time available. Test cases that cannot be executed do not add value to the customer.

- *Unnecessary / excess motion*. This waste is related to ergonomics. As testing is not so physical it might look difficult to relate this waste to testing. On the other hand we could translate this as the amount of releases to get to the required quality. The more (small) releases per period, the harder it might be to plan, prepare and execute the tests for these releases.

- *Defects*. Having a direct impact to the bottom line quality, defects resulting in rework or scrap are a tremendous cost to organizations. Here a distinction between defects in the testing process and defects in de product under test can be made. With testing we help the organization to eliminate the waste of product defects. Every product defect a test team finds can be fixed before going into production. This is probably the most recognizable waste related to testing. Defect in the test process itself is of course also a waste we have to be aware of. Improvement plans should also include our own test processes.

3. Six Sigma: improvement of processes and increase in the predictability of the outcome of the process

Six Sigma is a quality management method that offers a framework to manage quality. By many it is seen as a successor of Total Quality Management (TQM) with a high use of Statistical Process Control (SPC) as the underlying method. Processes can be controlled when you know how the flow of each process is and know what you have to measure. To measure is to know! Measuring is the basis of Six Sigma. The aim is to work smarter and get a higher quality. Sigma (∂) is the standard deviation from the average. It is a statistical term that measures how far a given process deviates from perfection. Six Sigma was founded at Motorola in the mid-eighties as a solution for problems with product quality and customer satisfaction. Six Sigma got its big popularity when it was used on a broad scale at General Electric and gained billions over a period of multiple years.

Six Sigma has four key elements:

- The customer is the starting point. Testing should focus on adding the best value to all stakeholders. E.g. set priorities based on a product risk analysis.
- Improve your test processes. Use models and methods as reference, but not as target. Use what suits best to the organization's own processes.
- Ensure teamwork. Not only within the test team but also with the other members within your projects.
- Collecting and analyzing data. Improving the test process cannot be done without measuring the effect of the changes made in the process. Keeping metrics, analyzing them and taking action should become a company's second nature.

4. Lean Test Management: managing the testing process to improve the quality of the test object to be measured within the planned time and budget

Within Lean Test Management the best of the Lean and Six Sigma theory is combined with testing. Within Lean the focus lies on eliminating activities that do not directly add value to the customer. (Lean) Six Sigma focuses on achieving a higher quality by working smarter.

Although the Lean method emanated from production and production related industries and Six Sigma is not specifically designed for testing, the principles are also excellent to use in testing processes. An example: within the testing process there are often delays because the testers are waiting for the designs or because the test environment does not meet expectations. By investigating this process, using existing techniques from Lean and Six Sigma, we can improve this process so no valuable time will be lost.

It is often tried to implement improvements to processes, which was not always successful. Why should it succeed when using Lean Test Management? Lean Test Management can be seen as a good addition to existing test improvement models. These models are based on a benchmark against which your organization is compared and focus mainly on the test process itself. The improvement techniques associated with Lean Test Management „talk" from your organization. The specific situation of an organization is thoroughly investigated and based on the results, the necessary improvements are considered. Not only testing is investigated, but also the surrounding processes and their interactions with the testing process are not forgotten. A lot of waste is often initiated on the interfaces of these processes and in order to make the combined process more "lean" it is a good idea to focus on those borders and get rid of that waste. There is much to be gained. Proven techniques and tools from Lean and Six Sigma will help achieving your improvement goals.

5. A roadmap to Lean Test Management

We think the best way to start implementing Lean Test Management is to start with the first phase of the Six Sigma DMAIC cycle. DMAIC is an extended version of the Deming cycle (plan, do, check, act). In addition it has a more important place for measurements and control. The following phases are part of the DMAIC cycle:

- **D**efine, to agree on what the (improvement) project is. Make sure to involve people from the whole process and not only testers to avoid sub-optimization. Sometimes the discussion between these people already reveals quick wins in the process because they are better aware of what is expected from each other. Typical Lean / Six Sigma techniques that can be applied here are SIPOC and Value Stream[1].

- **M**easure, evaluate the existing measurement system, observe the process, gather data and map the process in more depth. The areas of improvement defined in the first step already give an indication of problem areas in the process. It is important to know what the most important causes of

1 *The techniques used in this article are picked by the authors as useful for testing. Within Lean and Six Sigma a large amount of other techniques are available that you could use.*

these problem areas are, therefore we need to measure. The 20-80 rule applies here. Solving 20% of the most important causes might give you an 80% improvement in the problem area. A typical Lean / Six Sigma technique that can be applied here is the Pareto chart.

- **A**nalyze, use collected data to confirm the source of delays, waste, and poor quality. After the problem areas in the process have been measured, the data to find the real source of the problem can be analyzed. It is important not to skip this phase, even when you think solutions are obvious after the Define and Measure stages. You need to investigate deeper in order to come to the best solutions in the end. Typical Lean / Six Sigma techniques that can be applied here are the 5 Why's and Ishikawa diagrams.

- **I**mprove, to make changes in a process that eliminate defects, waste, cost, etc., which are linked to the customer needs identified in the Define phase. The Analyze phase made the waste in the processes clear. Now it is time to get rid of that waste by improving the process. Use existing test process improvement models and test methods as reference, but not as target. Use what suits best to your own organization and processes. A typical Lean / Six Sigma technique that can be applied here is the Pick Chart.

- **C**ontrol, to make sure that any gains a team makes last. Use the same measures as used in the Measure phase to monitor the processes and make sure that there are fewer deviations from the optimal process. A typical Lean / Six Sigma techniques that can be applied here is the Control Chart.

As indicated in each phase described, there are a lot of ready to use techniques available that can help you to start improving right away. If you want to know more about the techniques, the references at the end of this article are a good start.

6. Conclusion

After describing the principles of Lean and Six Sigma and how they can be applied to testing, the authors hope that you will also have a good feeling that combining these methods will lead us into an efficient, cost effective and high quality future.

Summarizing, by applying Lean Test Management,

- the test process will become more efficient. All activities that have no direct added value for the customer have to be deleted;
- the quality of products will increase by working smarter;
- test process improvement will start from within the organization and existing processes, and therefore the best tailored results can be obtained;
- insight will be gained into the processes around testing that also need improvement in order to make the testing process more efficient.

References:

- Lean Six Sigma, Michael L. George, ISBN: 9780071385213
- Velocity, Dee Jacob, Suzan Bergland and Jeff Cox, ISBN:9781439158920
- Lean Thinking, James P. Womack, Daniel T. Jones, ISBN:9780743231640

> biography

Bob van de Burgt
is test consultant at Professional Testing B.V. He contributed to the development of the testing method TestFrame® and the test management approach of Logica for which he also was co-author of the published books. Bob has given many testing courses (including ISTQB) and is a frequent speaker at (inter) national congresses. He has been the chairman of the Dutch Special Interest Group in Software Testing TestNet for many years and was EuroSTAR Programme Chair in 2008.

Iris Pinkster-O'Riordain
is test consultant at Professional Testing B.V. and has experience in testing and test management since 1996. She co-developed Logica's method for structured testing: TestFrame ®, their test management approach (Risk & Requirement Based Testing) and TestGrip, the method on test policy and test organization. She is co-author of the books published on these topics. She often speaks at (inter)national congresses. In 2007 she won the EuroSTAR award for "Best Tutorial".

Collaboration Equals Quality

by Debra Forsyth & Dave Lloyd

We have all been there. The project schedule is laid out, six months to develop, and one month to test. Then the schedule slips and now we have two weeks to do all the testing. We know it's not the right way to build software, but what is? How can we ensure the quality of a product that meets the stakeholder's needs in an acceptable timeframe? The future of testing will be to build multiple quality indicators into the process so that when the development is complete, we are able to measure the quality of the application.

Although many teams utilize some of the ideas we will discuss, only a team that collaborates to implement multiple quality indicators throughout the project will have complete confidence in their product's quality. We are all responsible for quality, not just the QA team.

The tools to aide teams in this endeavour are available today. What is required in the future of testing, however, is a change in attitude and the division of responsibility.

This article was written by two people: A tester and a developer. Two people who understand and have experienced the importance of collaboration and multiple quality indicators in the software development lifecycle.

This article will look at the following quality indicators that are implemented by the whole team throughout the project.

- Requirements Reviews / Story Boarding
- Architecture Validation
- Static Code Analysis
- Code Reviews
- Paired Unit Testing / Tester & Developer
- Continuous Integration
- Test Impact Analysis
- Code Coverage
- Exploratory Testing
- Stakeholder Feedback

With the above quality indicators in place the test cases left to execute prior to implementation are minimal and more concentrated on integration, performance, security and usability. Throughout this article you will see the words "the team" or "teams". The team includes the roles of developers, testers, business analysts/product owners, architects and graphical designers. A team is made up of the roles that are required to produce the application under development.

Requirements, User Story Reviews and Story Boarding

Understanding what the users are asking for and then laying out those needs as requirements is the most complex part of an IT project. How do you know that the requirements documented are what the users are asking for? Have the business and/or users really expressed what they need? What people hear and read is processed differently to the point that two people can and do take away completely different meanings of the same information. Add in different learning skills that are not always taken into account in the business world, we end up with requirements being misinterpreted to some degree. In today's global marketplace, we have teams that speak multiple languages adding more complexity to the process. Add to all that the fact that requirements are dynamic - they can change at any time.

There are requirements that are not outlined in user needs. These items are known as Quality of Service (QOS). During the design review, requirements for the following have to be reviewed and their need established.

- Security and Privacy
- Performance
- User experience
- Manageability
- Hardware

Business needs and importance have to be established and added to the requirements as a priority. Establishing requirement priority is difficult and getting it right is not always thought of as important.

User satisfaction is another issue that needs to be understood

Belgium Testing Days

March 12–14, 2012
Brussels, Belgium

www.belgiumtestingdays.com

Belgium Testing Days

Tutorials – March 12, 2012

"Influence and Authority: Using Your Personal Power to Get Things Done"
Johanna Rothman
09:00-17:30

"Making Test Automation Work in Agile Projects"
Lisa Crispin
09:00-17:30

"Test Estimation, Monitoring and Control"
Lloyd Roden
09:00-17:30

"Mobile Testing"
Karen N. Johnson
09:00-17:30

"Essential Software Requirements"
Lee Copeland
09:00-17:30

All tutorials include lunch and coffee breaks.

Conference (Day 1) – March 13, 2012

Time	Galaxy 1	Galaxy 2	Galaxy 3	Workshops -Atrium	Sponsor Tracks
08:00-09:00	Registration				
09:00-09:15	Conference Opening				
09:15-10:05	**Keynote** **Goranka Bjedov:** "The Future of Quality"				
10:10-11:00	**Kris Laes:** "When performance testing meets Business people"	**Maarten Van Eyken:** ",QA'-gile: black-box on a white-board"	**Johan Jonasson:** "Don't Mislead Your Stakeholders (Even If They Ask You To)"	**Dorothy Graham & Mark Fewster:** "Test Automation Clinic" - Part 1 -	*Sponsor Presenter*
11:00-11:25	Coffee Break				
11:30-12:20	**Michel Kalis:** "Perfomance Testing Case Studies"	**Henrik Andersson:** "Hi, are you stuck in an agile project?"	**Susan Windsor:** "How to deliver value from Test Assurance"	**Dorothy Graham & Mark Fewster:** "Test Automation Clinic" - Part 2 -	*Sponsor Presenter*
12:20-13:50	Lunch				
13:50-14:40	**Sajjad Malang & Catherine Decrocq:** "ATDD with Robot framework done right"	**Elalami Lafkih:** "Testing Serendipity Testing: The Art of Increasing Defect Detection Likelihood"	**Niels Malotaux:** "Quality Comes Not From Testing"	**Dawn Haynes:** "The Search for Software Robustness" - Part 1 -	*Sponsor Presenter*
14:50-15:40	**Michael Palotas & Dominik Dary:** "Test Automation – 10 (sometimes painful) Lessons Learned"	**Matthew G. Sullivan:** "Would You Enjoy Reading Your Own Test Reports?"	**Jean-Paul Varwijk:** "Regulations – Where Quality Assurance meets Testing"	**Dawn Haynes:** "The Search for Software Robustness" - Part 2 -	*Sponsor Presenter*
15:50-16:20	Coffee Break				
16:20-17:10	**Keynote** **Johanna Rothman:** "QA or Test? Does it Matter? You Bet it Does!"				
17:15-18:10	**Lightning Talks** Speakers: Lisa Crispin, Scott Barber, Dawn Haynes, Dorothy Graham, Susan Windsor, Lee Copeland				
18:10-18:20/ 18:30-19:20	Cocktail / Improvisation act				
19:20- 22:30	Dinner & Chill Out				

Conference (Day 2) – March 14, 2012

Time	Galaxy 1	Galaxy 2	Galaxy 3	Workshops -Atrium	Sponsor Tracks
08:00-09:00	Registration				
09:00-09:05	Conference Infos				
09:05-09:55	**Keynote** **Karen N. Johnson:** "Why it matters what I'm called: Quality Analyst or Software Tester"				
10:05-10:55	**Scott Barber:** "Applying Educational Assessment to Testing"	**Gilles Mantel:** "Test automation: the return on investment myth"	**Dries Baert:** "Offshore tester: Friend or Foe?"	**Eveliina Vuolli & Kirsi Korhonen:** "Solving practical problems with the quality assurance in the large scale" - Part 1 -	*Sponsor Presenter*
10:55-11.25	Coffee Break				
11:30-12:20	**Peter Morgan:** "Planning your career to stay testing into the sunset ..."	**Wim Demey:** "Knock-knock-knockin' on infrastructure's doors"	**George Wilkinson:** "Creating balance as a tester in modern times"	**Eveliina Vuolli & Kirsi Korhonen:** "Solving practical problems with the quality assurance in the large scale" - Part 2 -	*Sponsor Presenter*
12:20-13:50	Lunch				
13:50-14:40	**Raja Bavani:** "Distributed Agile: The Need for QA Mindset in Agile Testing Teams"	**Graham Thomas:** "Test Process Improvement – Answering the BIG questions!"	**Stefaan Luckermans:** "Janssen of Janssens, Thomson or Thompson, Dupont ou Dupond?"	**Goranka Bjedov:** "Advanced Hands-on Performance Testing" - Part 1 -	*Sponsor Presenter*
14:50-15:40	**Björn Vanhove:** "Think out of the box with ADQA"	**Rik Marselis:** "Governance: Controlling quality like a filmdirector"	**Adrian Rapan & Tony Bruce:** "A Tale Of Two Cities"	**Goranka Bjedov:** "Advanced Hands-on Performance Testing" - Part 2 -	*Sponsor Presenter*
15:40-16:10	Coffee Break				
16:10-17:00	**Alfonso Nocelli:** "Open Source or not open Source that is the Question"	**Sigge Birgisson:** "Moving the project forward - perform testing and avoid the QA"	**Zeger van Hese:** "Artfull Testing"		
17:05-17:55	**Keynote** **Lloyd Roden:** "Dispelling Testing's Top Ten Myths and Illusions"				
18:00-18:10	Closing Session				

Note that the program is subject to change. Please visit our website for the current program.

Belgium Testing Days

March 12-14, 2012
in Brussels, Belgium

QA versus Testing! Antagonism or Symbiosis?

is next year's theme of the **Belgium Testing Days** taking place from March 12–14, 2012 in the Sheraton Brussels Airport Hotel in Brussels, Belgium.
The Belgium Testing Days is an annual European conference for and by both national and international professionals involved in the world of Software Testing.
Learn from experts and many others who are passionate about Testing during 3 days of talks, learning and discussion and use networking opportunities with your peers and industry experts.

Belgium Testing Days 2012 –
A Díaz & Hilterscheid Conference
Endorsed by AQIS

info@belgiumtestingdays.com
www.belgiumtestingdays.com

Follow us on Twitter: www.twitter.com/BelgiumTD

Register Now!
Catch the Early Bird discount and save 15% of the regular price!

Impressions and quotes from 2011


Knowledge Transfer


Networking

"It was a real **informative** and **productive** conference."

"It gave me a lot of **new inspiration**."

"I'd like to have this **yearly**."

"**Good atmosphere** and facilities and **well planned** and **organized**."


Exhibition


Entertainment

Want to exhibit?

If you'd like to be an exhibitor at Belgium Testing Days 2012, please fill in the form which you can find in our exhibitor brochure and fax it to +49 (0)30 74 76 28-99 or e-mail it to info@belgiumtestingdays.com.

Download our exhibitor brochure at www.belgiumtestingdays.com

Exhibitors & Supporters 2012

coralius · bcs The Chartered Institute for IT · ps_testware Your devil's advocate · quasus · MICRO FOCUS · SOGETI · te testing experience · Agile Record · Díaz Hilterscheic · AQIS

Díaz & Hilterscheid Unternehmensberatung GmbH

Kurfürstendamm 179
10707 Berlin (Germany)
Phone: +49 (0)30 74 76 28-0
Fax: +49 (0)30 74 76 28-99
www.diazhilterscheid.de

AQIS bvba

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

up front. The KANO Analysis (http://en.wikipedia.org/wiki/Kano_model) is an interesting concept that establishes exciters, satisfiers and dissatisfiers. Satisfiers are requirements that achieve the user's goals and make the product enjoyable to use. Dissatisfiers are requirements that if absent you will hear about, but are not always identified up front. The performance QOS is a dissatisfier. Performance is not always established by users, but when the product is slow you will hear about it loud and clear. Exciters are requirements that add a "wow" effect for the user. Knowing which KANO the requirements fit into will help in establishing user satisfaction.

Story boarding tools on the market today are very powerful, allowing for the design of the user interface (UI) views and detailed interaction flow. In many tools, the storyboarded UI can have actual working features embedded. You can test that related requirements interact as a coherent experience. This allows the users and the team to see some of the requirements visually, and even interact with them. Story boarding the requirements will return actionable feedback from the creators and stakeholders, identifying missed and flawed requirements, and misinterpretations. Your storyboard can be used to review what is about to be developed with all team members right up to company management. This is your chance to show off what the team is about to create and resolve any issues that are identified. This is the most important early quality indicator the team has.

Figure 1 is an example of a storyboard for a company that requires an end-user self-service ticket feature and the ability to track technicians. Microsoft has not yet released the Storyboard tool.

Figure 1: Microsoft Storyboarding tool

The **Requirement Quality Indicator** is needed to confirm we are about to develop what the user wants and what the business needs. Through requirements gathering, requirements review, design review, storyboarding design and review, and user satisfaction review you are testing and able to identify the quality of these items. If the quality indicator is not satisfied, the team will have identified any issues and addressed them or established a risk. It is an ongoing process that requires the whole team's participation. Knowing your requirements have been verified early in the development cycle, you are no longer finding out later in the process that there is a problem.

Architecture Validation

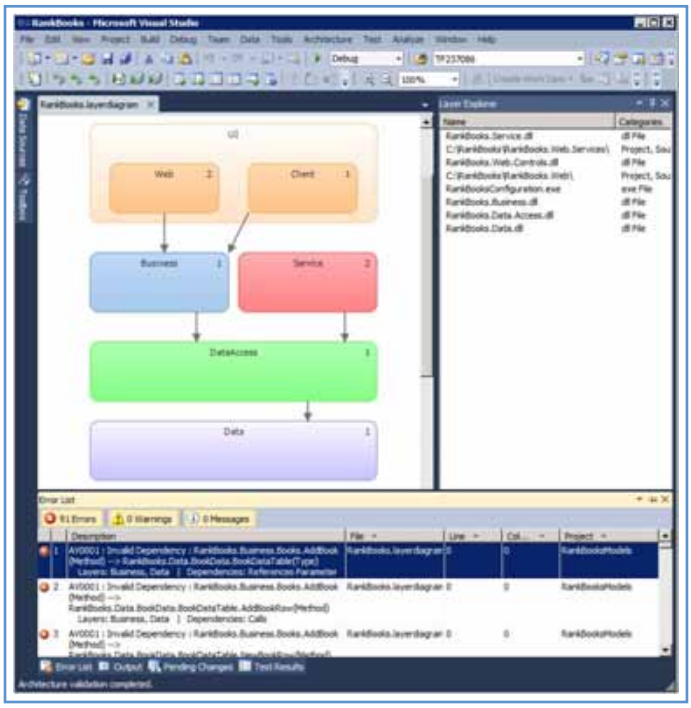


Figure 2: Architecture Validation in Visual Studio

A solution architect lays out a framework that helps the team build an application with well-defined layers. This is done for many reasons; the application modules have a clear separation of concern, and are therefore more maintainable and easier for someone new to the project to follow and understand. It also has the benefit of making it easier to swap out one technology for another. For example if your data layer is properly decoupled from the other layers, then you should be able to swap out Oracle for SQL Server with a minimum of disruption. Over time, this well-meaning architecture often slips away. As developers are pressed for time, they start to make mistakes. It's faster to call the data access layer directly from the user interface to get that one tidbit of data, than it is to have to create a class for the business layer. What if you could create a layer diagram, drag your source code into it and validate whether the developers are following the architecture or not. Unchecked, these issues will make the application less maintainable over time, while catching them right away makes them much easier to fix.

Static Code Analysis

Many development teams do not put enough stock in following coding standards. These standards can save you a lot of time and money in the long run. Static code analysis is an analysis of your code without running the application. Its purpose is to point out code that does not follow predefined coding standards. A Static Code Analysis tool will scan your code looking to see if certain pre-defined rules are broken. Many bugs can be traced back to the breaking of a simple coding rule. For example, if you are writing an application that must be translated to other languages, you must not include a hard-coded string. All text displayed to the user must come from a data source of some kind so that the application can determine the language of the user and retrieve and display the string properly. There are many such rules that should be followed: rules for security, interoperability, mobility, and performance and so on. Running your code through a Static Code Analysis tool will undoubtedly catch some problems that would have led to bugs in the completed version. There are many tools that do this kind of evaluation. If your team uses Visual Studio to write code, the Static Code Analysis tool is built right into the product. Each project in a developer's solution can have specific rules turned on that will generate warnings or errors during compile. The automated build process in the team foundation server can be set to perform Static Code Analysis on any project that it builds. This way if a developer forgets to run it on their own code, the build will catch it and generate the warning. Catching these kinds of issues early on will save the team a lot of time during their project.

Code Reviews

Earlier in the article we mentioned the importance of design reviews. Reviews are important throughout the application lifecycle to ensure everyone has a complete understanding of, and input into, the decisions being made. Code reviews are no different, the developers in a team should have all code reviewed by a peer. This should be treated as an opportunity to learn. That goes both ways...the reviewee often learns from the reviewer. A code review not only helps to ensure the code is well written and maintainable, it also allows the developer to explain out loud what they are trying to accomplish. This will help them to understand the problem and get it clear in their mind. The code review should not be too formal a process. A good development team will do code reviews naturally by talking out tough problems and sharing techniques and patterns with each other.

Code reviews will catch a lot of misunderstandings in the requirements. As developers have to explain to each other what they are coding, problems with the requirements are sure to come out. This is a much less expensive time than during traditional testing, stakeholder reviews or, worse, in production.

Make the code review mandatory but try to keep it light. Let team members review each other's code in an informal and fun learning environment.

Paired Unit Testing – Tester & Developer

Microsoft states that the primary goal of unit testing is to take the smallest piece of testable software in the application, isolate it from the remainder of the code, and determine whether it behaves exactly as you expect. Each unit is tested separately before integrating them into modules to test the interfaces between modules. Unit testing has proven its value in that a large percentage of defects are identified during its use. Over the last few years some developers have adapted to Test Driven Development (TDD). The TDD approach is to:

1. Write a unit test that fails
2. Write just enough code that the unit test passes
3. Refactor the code, if needed
4. Execute the unit test again
5. Repeat until the requirement is fulfilled

The idea of creating more detailed unit tests that encompass many different passes through code is only just starting. In fact, there are teams that do no unit testing at all. Hard to believe, but true. As we all know, bugs cost more to fix the longer they are around. And fixing bugs has the potential of creating additional bugs. Bugs can become a ping-pong game between development and test which is a waste of time and effort for everyone. Being able to have cleaner code when development is done is one way to reduce the time playing ping-pong. Given the fact that developers are not testers and that they do not think like a tester, how do we get more out of unit testing?

When testers look at requirements, it is from the perspective of how will the users use this versus the developer's approach of how do I code this to make it work. Testers also use a "given/when/then" thought process to come up with the negative testing that will need to be executed. When you put all these approaches and processes together and create unit tests, you end up with very extensive testing early. The result is a lot fewer bugs to contend with later and better code earlier. Here is how we approach this scenario. A developer and tester pair up to write the unit tests. In pairing the tester and developer, we are obtaining a combined knowledge of what needs to be tested and how to write the tests. It is that simple and the effect is robust unit tests that cover most, if not all, of the requirement's functionality. The whole team knows what has been tested by the unit tests, which today does not often happen. The unit tests are automated and can be run against the build becoming regression tests. The pass and fail of the automated unit tests is a quality indicator. Having a suite of unit tests that become an automated build verification test (BVT) is another quality indicator. The BVT is continually looking for unknown side effects and errors due to changes in new code. It is regression testing the code and is a quality indicator used to determine whether the solution is ready for further testing.

It has been predicted that the future of testing will put more emphasis on testing early through unit tests with the involvement of testers. Some say testers need to learn how to code unit tests in the future. Pairing the teams is another way and gives us the quality indicators needed early.

Continuous Integration

You may have heard about continuous integration. For those that have not, let me explain the concept. When you have several developers on one team, they all work out of a central source code repository, or source control system. Let's say I have two developers Mike and Jason. Mike and Jason are both starting to work on new features today, so they both get the latest code from source control and update their local machine with that code. Mike makes a change to the code that adds a new customer to the system. The new feature requires that when adding a new customer, you must pass in a main contact. Meanwhile Jason is making a call

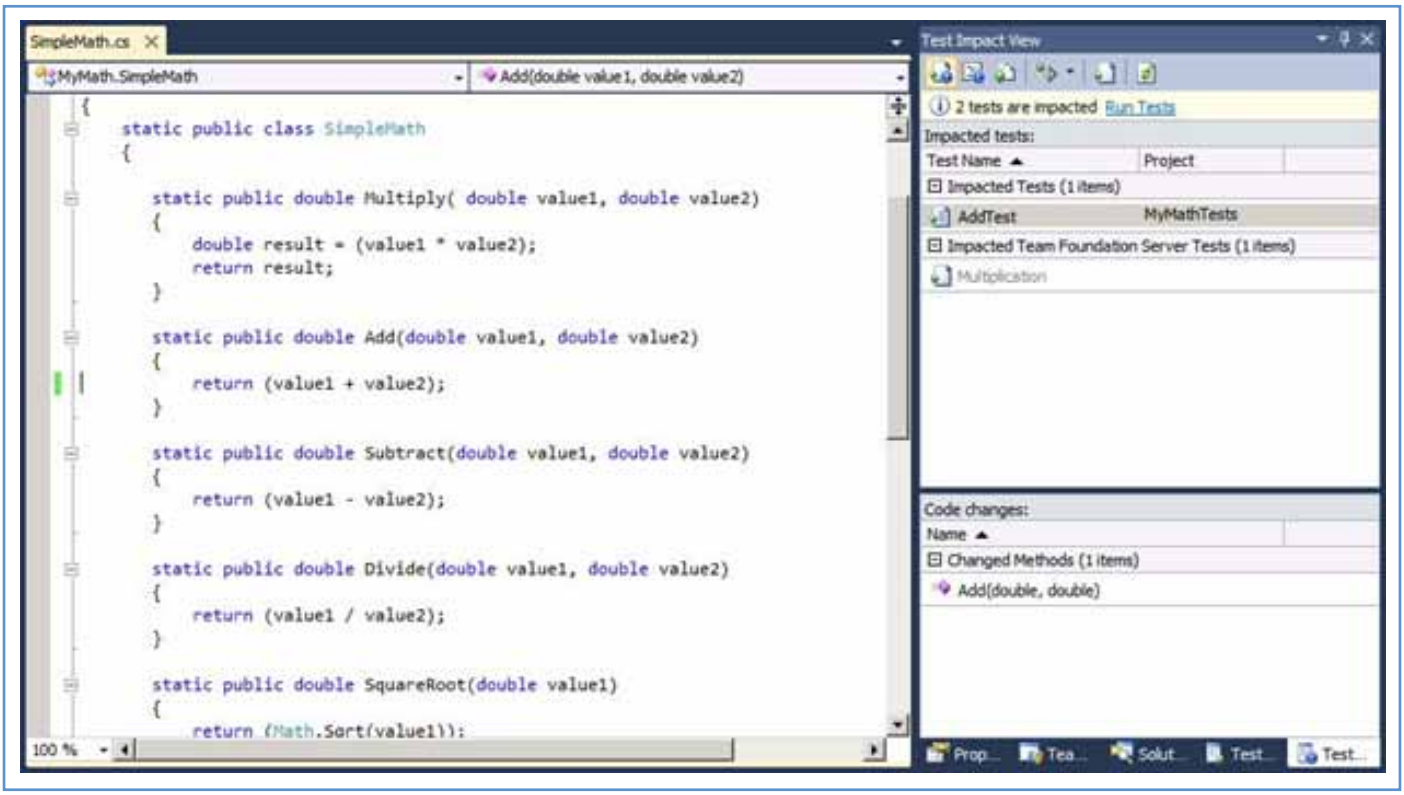


Figure 3: Test Impact - unit tests impacted by changed coded

to the new customer code from the new lead generation module. However, Jason doesn't know Mike has changed the interface used to add a customer.

Jason compiles his code, runs the unit tests he created, and validates that all is working fine. He is able to add a customer from the lead generation module. So he checks his code into the source control repository. Meanwhile Mike has done the same thing and is ready to check his code in, not knowing that Jason made a call to the Add Customer code without passing in a contact. Neither developer knows anything is wrong with the code base. Their local code still works fine, it compiles and all the unit tests pass. Several days later another developer, Alyssa, gets the latest code from source control so she can fix a bug. After getting the latest code, she finds out the code won't compile. Now Alyssa has to figure out what's wrong. Meanwhile it's been days, so neither Jason nor Mike assumes it's anything to do with them since everything still works fine on their local machine and has for days.

Continuous integration is the practice of building the code in the source repository every time someone checks in code. Therefore when Jason checked his code in, the build server would have retrieved the latest code from source control and compiled it, letting Jason and the rest of the team know Jason's code was integrated into the repository without issue. However, when Mike checks in, the build will fail, letting the team know that Mike checked in code that does not integrate with the rest. The good news is Mike finds out right away and he knows it has something to do with

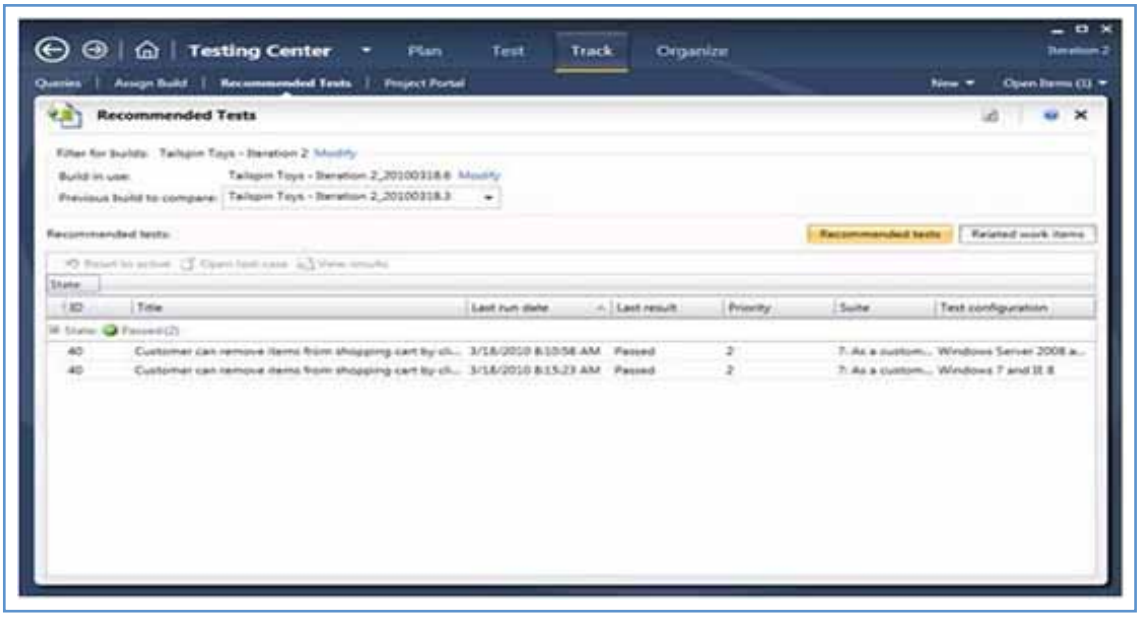


Figure 4: Microsoft Test Manager Recommended Tests

the code he just checked in. Therefore finding the problem and fixing it will be much faster for Mike than it would have been for poor Alyssa who was blind-sided.

Continuous integration is considered an agile practice; however, every team should implement it. It costs nothing, a build machine is doing all the work, therefore there is no interruption to the development team except to let them know what they checked in was integrated properly with everyone else's code.

Test Impact Analysis

Today's applications are complex to say the least. Test impact analysis addresses the issue of changes being made to code or the database that affect already tested requirements. Test impact will verify changes prior to check-ins by showing a list of changed methods in code. It compares the current build against the last successful build. Based on the methods and historical code coverage, the tool finds what test invoked the methods and their dependants and recommends those tests be executed again.

Not only does Test Impact notify the developer of impacted tests, but it also notifies the testers that test cases have been affected. In Test Manager there is a view called Recommended Tests that, with each new test build, will display any test cases affected by changes since the last build. Is this not the reason for regression testing - to find broken features that once worked fine? With automated regression unit testing and Test Impact, what more testing do we need to verify our solution is still working? Some additional testing may be needed, but not a lot.

Test Impact is our quality indicator police. It is continually watching for changes that might impact the quality and letting us know when something is found.

Code Coverage

100% code coverage seems like an unattainable goal. However, it is a great tool for letting the team know how much of the code is executed by their automated tests. Code coverage tools tell us what parts of the code are executed by our automated tests.

Tools like Visual Studio can monitor what code is executed when tests are run and then report back what parts of the application, from a code point of view, are not being executed, as well as the code that is being executed.

```
int x = 3, y = 4, z;

switch (x)
{
    case 0:
        y = 4;
        z = 2;
        break;
    case 3:
        if (y == 4)
            z = 3;
        else
            z = 2;
        break;
    default:
        break;
}
```

Figure 5 Code Coverage Coloring

Why would I say this is an unattainable goal? It's very difficult to achieve 100% code coverage. I have seen many teams attempt to reach this goal at the cost of good quality tests. The first priority should be quality tests on the most complex code. Use code coverage to point out areas that require unit tests and fill those in. Having a simple test for every line of code is not better than having good quality tests on the code that really needs it.

Other testing

We are doing a lot of testing in the early phases. We have quality indicators that are ensuring good requirements, clean code and unit tests, test impact, performance testing, and possibly a build process that will deploy clean code to the test lab. We have code coverage that tells us what has and has not been tested. Our quality indicators are doing a good job of tracking quality, and we know it is good or we would not be at this stage yet. Test cases are written before you have working software and therefore only cover what you anticipated. Unit tests will have covered a lot of your test cases and been checked off as done. The combination of all the early quality indicators have come together to direct you to what has not been tested. Once you have the solution in a test lab, the last quality indicators are:

- Test cases not covered by unit tests need to be executed
- Verification that the software meets the requirements from the user's perspective
- QOS testing that unit tests could not achieve
- Environmental compatibility with other software already deployed in production

Exploratory Testing

Verification that the software meets the requirements from the user's perspective and environmental compatibility can both be done following exploratory testing. Exploratory testing is done by selecting a business process that may include many features. The example below relates to shopping online:

1. Using IE9 browser, I open the shopping website.
2. I click on the link I think is correct and the site opens.
3. I browse the site looking at what is for sale; there are categories I can use to narrow down what I am looking for; I use them at times during my exploring.
4. I find the Login so that I can purchase what I've found.
5. I enter my user name and password and login.
6. I select my item to buy and check out.
7. I enter all the details for billing and shipping.
8. I submit my order.
9. I log out of the site.
10. I close my browser.

Hierarchy	Not Covered(Lines)	Not Covered(% Lines)	Covered(Lines)	Covered(% Lines)	Partially Covered(Lines)	Partially Covered(% Lines)
Julian@CTP 22/08/2004 16:17:08						
UnitTestDemo.dll	19		4		0	
UnitTestDemo.MathOps	0	0.00 %	4	100.00 %	0	0.00 %
UnitTestDemo.Properties.P	11	100.00 %	0	0.00 %	0	0.00 %
UnitTestDemo.Properties.S	8	100.00 %	0	0.00 %	0	0.00 %

Figure 6 Code Coverage

One Easy Tool for Testing

Your App's UI and Performance

Telerik Test Studio

Effortless Functional Testing

- Test web and desktop apps
- Next-generation test recording
- Record once, run against multiple browsers

End-to-end App Performance Testing

- Unparalleled insight into your app performance metrics
- Visualize performance stats over time
- Easily run a functional test as a performance test

Download your free trial and get 20% off Test Studio at:
www.telerik.com/TestingExperience

telerik
deliver more than expected

In this example we are testing many features in a system end-to-end format. We are essentially verifying that the software meets the requirements from the user's perspective. We are also verifying environmental compatibility. Like the scenario above, we would test different routes through the shopping online solution including different browsers. It has been said by many experts that exploratory testing discovers more bugs than structured testing. Exploratory testing can be a very valuable quality indicator. Two great experts on exploratory testing that are a must to read are Session Based Test Management by James Bach and Exploratory Testing Tours by James Whittaker.

Microsoft is introducing a new exploratory testing tool that will track the steps taken during testing and allow the tester to add comments related to specific issues found. Below is an example of how the tool looks during exploratory testing. During exploration, bugs can be created without stopping your testing. At the end of testing you can create a test case that will automatically include the steps taken. Often the exploratory test is required for regression testing.

In Figure 7 the tester has found that the "CreatedBy" and "AssignedTo" are blank when viewing the confirm deletion page. On the right panel they have described the issue and next will take a snapshot to add a picture to the panel. The last step will be to create a bug. Bugs such as these are rich with information for the developer to quickly analyze and fix the problem.

Stakeholder Feedback

Stakeholders are the people we are developing the solution for. We have engaged them during our quality indicator showing them how the solution will look, navigate and essentially work with the storyboarding tool. Now we want them to see the completed requirements. In an agile process, as each product backlog item is completed it is shown to the stakeholder for review as soon as the status has been set to done. If you are not explicitly using an agile methodology, there is no reason why you still cannot do this by requirement or groups of requirements. The feedback from the stakeholder is a quality indicator. Stakeholders may love what has been developed or they may suggest minor changes. If you have done the storyboarding, there should not be any major changes indicated during feedback.

How do we collect stakeholder feedback? The team demonstrates the feature in a production-like test lab, reviewing the requirement, explaining how it has been fulfilled and answering stake-

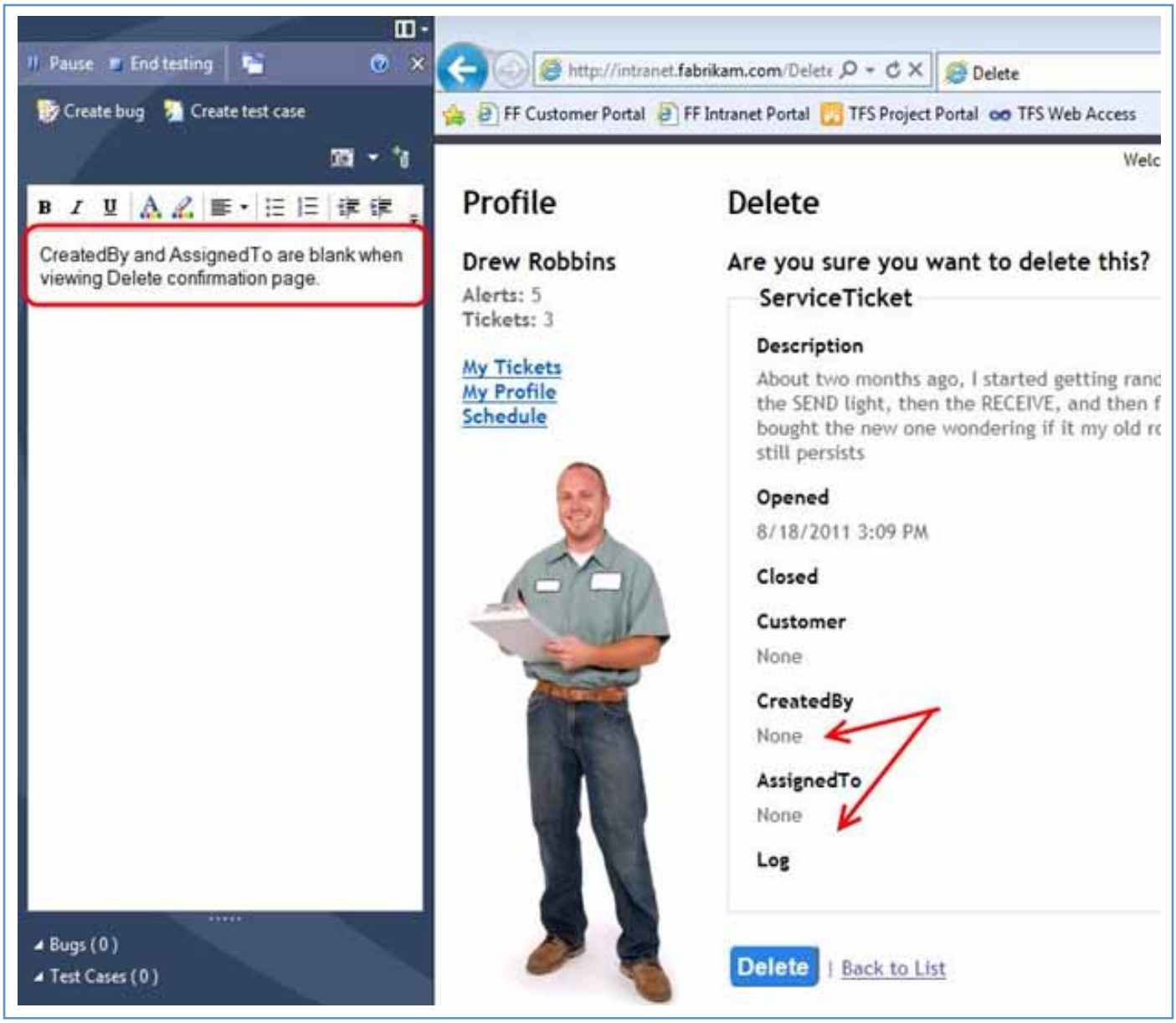


Figure 7: Microsoft exploratory testing tool not yet released.

Foto: HPI / K. Herschelmann

Potsdam

The most beautiful place for bright minds

Mr. Net is a Potsdamer! This interactive sculpture, created by the Spanish artist Jaume Plensa, is located on the grounds of the Hasso-Plattner-Institute, an unmatched centre of academic excellence for IT systems-engineering in Germany. The combination of first class alumni, innovative business start-ups, and successful companies makes Potsdam a premium IT location. Come to Potsdam and discover the city for yourself: as a lovely holiday destination, as a multi-faceted event location, or as an attractive place for your business.

Potsdam is a unique mixture of world heritage and high-tech, of science and business, of Brandenburgish cultural milieu and Mediterranean charm. It is an attractive city and a premium location for business with outstanding prospects for future growth. Potsdam is also a growing city with creative potential. Located right next door to the metropolis Berlin, Potsdam offers everything you need to succeed.
Potsdam: a great place for great ideas.

This great city welcomes you warmly!

POTSDAM

State Capital Potsdam
Department of Business Development
economy@potsdam.de I www.potsdam.de

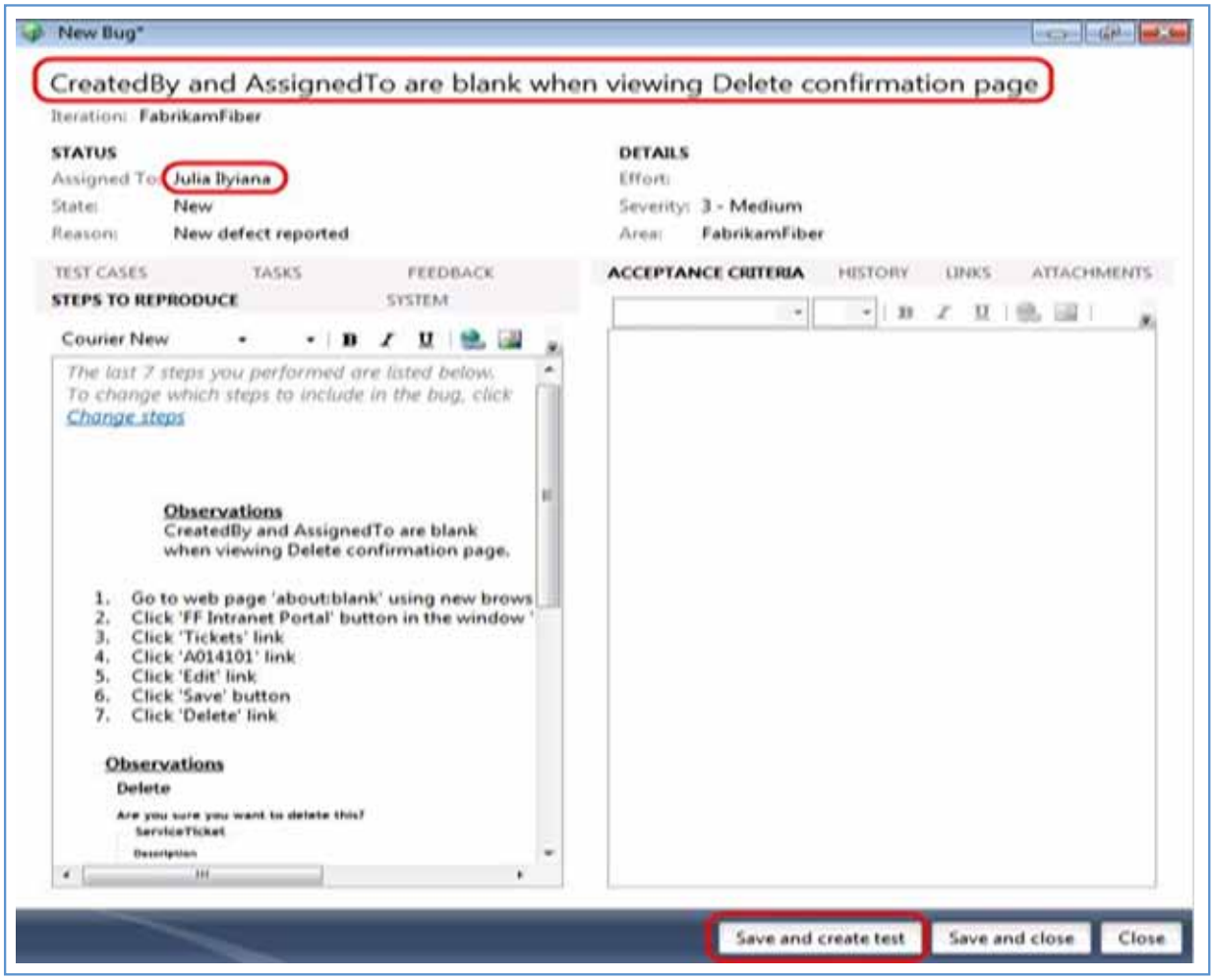


Figure 8: Microsoft Exploratory Bug

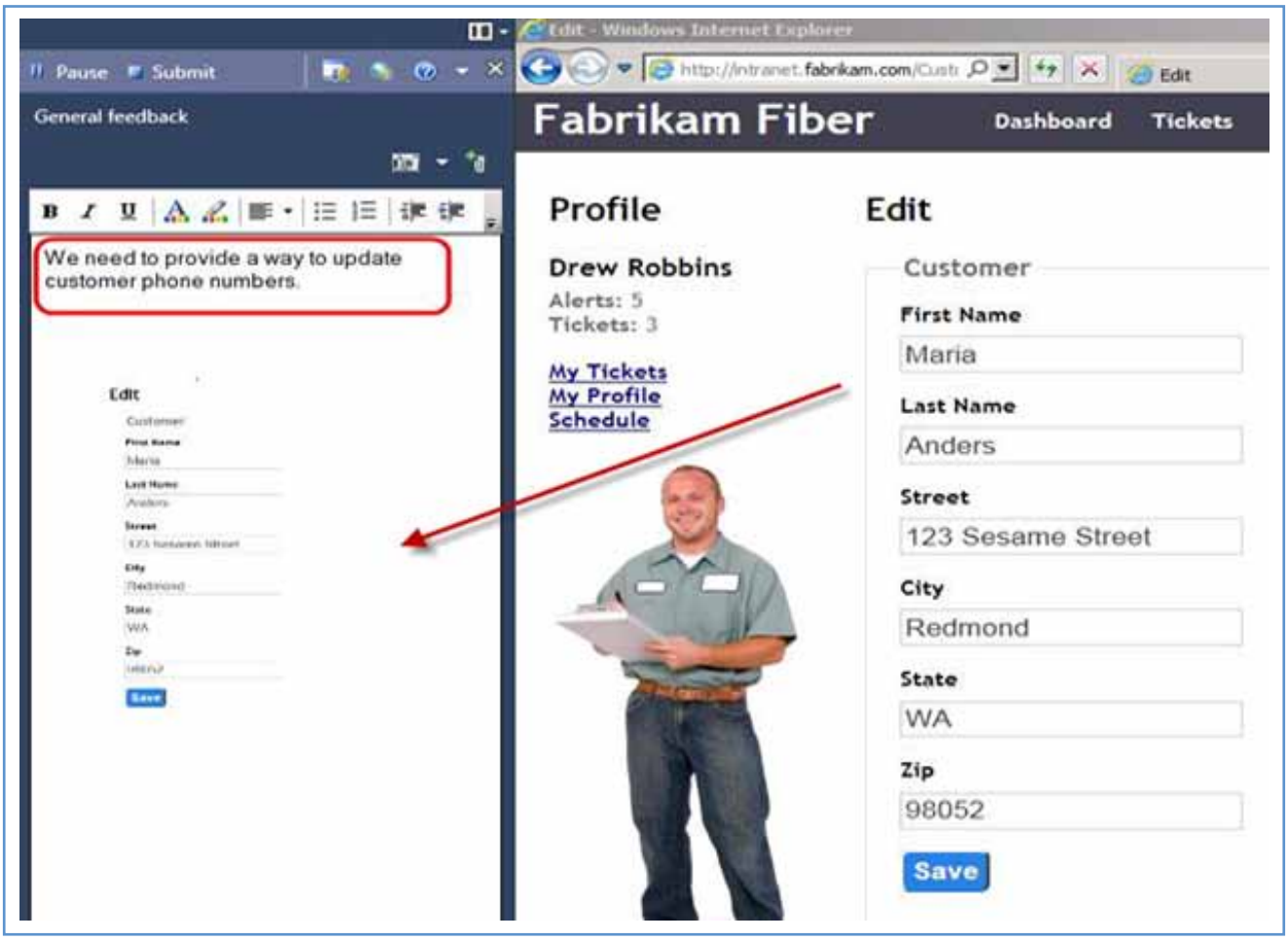


Figure 9: Microsoft Stakeholder feedback tool not yet released.

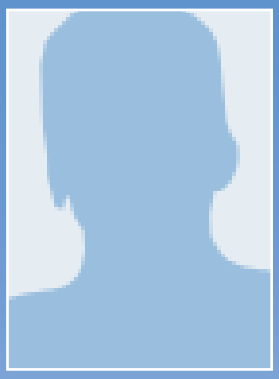
holder questions. The stakeholder can be the "driver" of this process or the team can be the lead. There are tools available that allow the stakeholder to do a review on their own and report back any issues or questions. Below is a preview of the Microsoft Stakeholder Feedback tool

Summary

The future of testing is changing along with the process of developing software. Agile and Scrum are breaking down the IT team silos and bring together all team members as one. The term 'development team' now includes both developers and testers. Separate team names are no longer used. We have worked in both a silo (as tester and developer) and in the new development team format. Our preference is the latter. We are still a tester and a developer and have not lost our identity or expertise. However, we have realized that working together to build a solution that meets the stakeholder's wants and needs is much more satisfying, less stressful and much more productive. As a team we can collaborate to ensure we are all working to achieve the same goals. As a team we can monitor the quality indicators outlined in this article easily and react quickly when the indicator is low. Using ALM tools like Microsoft's we are able to monitor all the quality indicators and at any time know the quality of what is being produced. When we collaborate, whether it be synchronously or asynchronously, we are able to share and view information and provide feedback at different points in time.

Collaboration definitely does equal quality.

> biography



Dave Lloyd
is a senior partner and co-founder of ObjectSharp. He has 25+ years' experience in the IT industry designing and building software solutions for a large number of clients in varying industries. Dave is a seasoned project manager with a great deal of experience implementing process into development teams, from small and large ISVs to in-house development teams. Dave has also spent time during his career implementing test solutions for clients, working with the most current automated test tools and implementing successful test environments. Dave brings to the table 20+ years of teaching experience.
Because of Dave's experience in the many roles of a software development team, it's only natural that he would gravitate toward Team Foundation Server and the ALM (Application Lifecycle Management) world.
A lot of Dave's assignments involve helping teams with their development process and to better utilize Team Foundation Servers to enhance their software development lifecycle. Dave has received the Microsoft MVP award for Team System.

Debra Forsyth
is the Quality Assurance Practice Lead and a Senior QA Consultant with ObjectSharp Consulting. As QA Practice Lead, Debra is an experienced Visual Studio Team System and Test Edition instructor and consultant providing training and support to clients. Recently Microsoft awarded Debra the MVP (Microsoft Most Valued Professional) award for her involvement with the community. Debra is the first person to be awarded an MVP for their involvement in the testing community, as well as the first woman to be awarded the ALM MVP (Application Life Management). She has over 25 years' experience in software testing within various business fields. A passionate advocate of the Agile/Scrum methodology, Debra mentors companies and test teams in the process of becoming Agile.

Skilled in the design and execution of comprehensive Software Quality Assurance processes/procedures, with proven qualifications in test life cycle, strategy/plan, test case/data design, risk assessment, functional/performance automated tools implementation, environment set up, defect and configuration management, Debra has succeeded in helping many teams turn their testing effort into a successful endeavor that raised the product quality and met project time lines. As well, Debra has a Computer Programming Certificate from Sheridan College, and is experienced with many different testing tools on the market today. She currently participates in Microsoft's Inner Circle meetings for all Microsoft testing tools. She is also the author and trainer for the ObjectSharp Team Foundation Server for QA Professionals course.
Her specialties: Microsoft VSTS 2010 for Testers, Test Manager, Coded UI, Web Performance and Load Testing; mentoring, training, software quality assurance best practices, Agile & Scrum methodologies.
Debra's published articles include the 'Big Bang Theory' in Agile Record.
Follow our blogs at:
Testa's Paradise: http://blogs.objectsharp.com/cs/blogs/deb/
Dave's 2 Cents: http://blogs.objectsharp.com/cs/blogs/dave/

The Future of Testing is Bleak

by Peter Russell

What is Testing

Over the last 30 years we have seen both the birth of software quality and its evolution as a key methodology. Throughout this evolution we have failed as an industry to recognize software development as a commercial activity that pays little attention to quality.

Testing software, or the higher idea of software quality, is foreign to most people even in the software industry. To the uninitiated it may seem more akin to placing the "QA Passed" sticker on a widget as it leaves an assembly line. In fact, software testing is very different to this naïve view. When testing software, each widget (component + version) we examine is new, and each widget does something slightly different to the last. Imagine that you are the tester, and you see a new kind of funny widget. How do you test it? How is it supposed to work? These are the same questions that software testers ask. The widgets are different each time because software keeps changing. This is why the versioning of software is so critically important. Testers do not occupy their time testing and then retesting the same version over and over again, they test a new version each time.

Regression testing verifies that the parts of the system that were not intended to change have not changed. This is different from the new widget model. Confusion between quality, testing and regression is common. The confusion shows in testing strategies where plans and resources are misaligned to outcomes.

- The entire working of the factory is a quality process.
- Validating a particular component + version combination is testing.
- Detecting uncontrolled changes is regression.

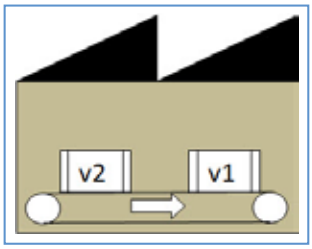


Figure 1 – Quality, Testing and Regression

Why is Testing Devalued

Management tell themselves many stories why testing is not a good commercial investment. It's a bit like going to the dentist, everyone knows they have to do it but no one wants the pain. Business software is rarely procured based upon its quality. This is simply because vendors don't know what quality is, or how to measure it, and customers just aren't asking the right questions. I've been involved in several product selection panels, and in every one it was never a criteria that the candidate software be of any specified demonstrable quality. Business people focus on features not on quality, and the main thrust of most evaluations is feature matching. We ask the question, "Can we live with this feature?", "Can we map our business process to this feature constraint?", but always with the naïve assumption that the software does what the salesman says it does, or what the manual implies.

There is a pervasive myth that "Feature Content" is more important than "Feature Working". The root of this myth is grounded in the manufacturing processes and quality control techniques that were inherited from the industrial revolution. The development of the quality mindset began with craftsmen, where no two artefacts were the same and the focus was on the man because the process was a highly prized secret. The industrial revolution then focused on repeatability, process designers are the great engineers of the day. This era also created quality control and the idea of tolerance. To improve yield, management widened acceptable tolerance. A high quality product was one with small tolerances, tight fitting and a high finish.

The idea of tolerance is still used today with software. We can measure it by tracking defects and coverage, and express it through exit criteria. When management wants to improve yield or bring a release to market by a given date, the option is available to vary the release exit criteria, thus changing the acceptable tolerance. When budget priorities are considered, tolerances widen and so tolerance measurement itself becomes less valued. The separation of the development and testing groups reinforces the idea that developers create useful things, but testers only test them. In a pinch, testers are the first to suffer based upon the view that their contribution is "more optional" than that of developers.

To apply a tolerance based scheme to software exit criteria means

having to keep good records. In this way you have something to measure, rate and prioritize. Without defect tracking based on a known triage process and coverage metrics, defect statistics are meaningless. This is why exit criteria tend to be motherhood and aspirational statements, with words such as "Testing complete" and "No high priority defects". But, what does it all mean? Testing complete just means you ran what scripts you had, and no high priority defects depends on what high priority means, and how much of the system was tested at all.

The great failure of software testing is the inability of test management to articulate why all this software metrics based complexity actually helps. Too many people believe that it does not, not because it's not true but because software failure risk is low, and customer "quality" sensitivity is low.

The Cost of Poor Quality

Software quality covers all the lifecycle processes, from methodology, project management, requirements management, configuration management, support and release processes. Testing is the bit you do at the end, poorly resourced and no-one's too sure what the functionality is anyway. Testing needs to be a mechanically comparative, traceable and auditable activity based on approved upstream requirements with the goal of reducing rework and support costs. Management have traditionally seen developer's time as an investment because they actually produce something, but unfortunately also see quality processes as costs. To counter this, we have to start factoring in the cost of support as the cost of development gone wrong, and the cost of quality processes as a multiplier to generate rework based cost savings.

Another example is the cost of customer implementation. In many cases implementation is charged separately to a customer, which means that customers pay for the vendor's failures through delays. When implementing fixed price, designers and developers risk having to bear some of the responsibility of poor implementation support features. Customer User Acceptance testing (UAT) is another area that can delay implementations, delay payment of invoices, risk missing contract milestones and invoke penalties. Every UAT issue is a failure of design and development, as developers have the resources and brief to deliver the whole system, the test resources do not.

Testing Tools

The only tools that have made any impact on testing in the last twenty years are the test automation tools and coverage analysis tools. The scripted automation systems have the power to create cost savings and introduce efficiency to test development and execution. In themselves, these tools are not a cure for the ills of software quality. Test automation brings with it a power that is keenly felt by developers when it is managed properly. This is the ability to change programmer behavior.

Programmers are given a great deal of discretion when it comes to how they spend their time developing, the degree of unit testing and how to prioritize deadlines. The high degree of creativity in programming makes for a typically low supervision, high trust environment. When an under resourced test group can barely scratch the surface of an application and is continually bound up with manual testing and keeping track of what is changing, programmers as individuals have little fear of discovery for small failures. As a consequence, a growing number of small errors compound in the application over time creating a large quality debt.

Test automation frees the testers from the time consuming drudgery of doing everything by hand. As a consequence, time is able to be invested in developing new feature coverage, thereby catching more small errors before release. The impact on developers is that now the probability of being caught out in small errors increases and more time is spent unit testing and double checking to avoid them. Voluntary participation in peer review processes also increases. No programmer wants to be thought of by their peers as the buggiest, and so the emotional impact of test automation should not be overlooked.

Coverage Analysis

The ability of a test manager to bring one hundred scripts into play at the end of release is meaningless unless the coverage of the scripts it# is understood. One hundred scripts covering 10% of the features is nothing to be proud of. If these scripts are also all manually executed, chronically out of date and difficult to maintain in an unversioned document format, then I would venture to say that testing has failed before it has started.

Coverage calculations are not easy but very worthwhile. In fact, management should be expressing testing and quality goals in terms of coverage. Consider the three main types of coverage: requirements coverage, feature or specification coverage and code coverage. The last is usually expressed as an edge coverage percentage. An edge coverage benchmark for an automated functional regression suite is doing pretty well at 60% and very well at 80%. When edge coverage is used in conjunction with unit testing, a unit test exit goal of 90% is not uncommon. Not all languages support easy code coverage analysis, but if yours does, this is very worthy of investigation. The other types of coverage are essentially non-technical and only require decent project management and record keeping to maintain traceability.

Are Games Different?

Games and gamers are different for the following reasons: 1) Gamers are serial purchasers, not one single large procurement; 2) They seek community advice and experience before buying; 3) They are not afraid to talk about bad experiences publicly. On the other hand, commercial software procurement focuses on: 1) Commercial secrecy surrounding a large IT investment; 2) A direct selling approach by vendors designed to isolate and control the customer experience; 3) Vendor protection from community knowledge of failed projects via mutual embarrassment. I use this example to underscore the idea that customers get the quality they want. If you compare a CEO and his 14 year old teenager, I'm sure his teenager would be shocked and dismayed at the lack of quality in their parent's business software compared to the community outrage brought about by defects in popular games.

Software Quality Checklist

Software quality is multi-dimensional. It's not just "one" thing you can score, and it can get complex. Here are some example questions:

- What are the exit criteria for the release of the software?
- Does your development methodology include keeping up-to-date approved specifications?
- Do you maintain test management records, e.g. cases and results?
- Do you measure traceability to a) requirements, and b) specifications?
- Do you do coverage analysis? What goals, and what results?

- Do you practice configuration management? a) What is the component architecture, and b) How are the components versioned?
- How do you measure a) Reliability, b) Availability, c) Serviceability, d) Maintainability?
- What are your Service Management SLAs?

The Future of Testing

I believe software testing is headed for darker days to come with further political isolation from management, less funding and poorer quality all round. The hopes of software quality theory and new testing technologies in the 1990s have been usurped by the likes of development focused ideas such as "Agile", "Object Oriented", "Cloud" and the $0.99 "Mobile" app. New languages and development methods are designed to allow developers to churn code out quicker, not for better review, maintenance, testing, traceability or audit. Maintenance and development costs will rise, testing budgets will fall, and more projects will fail. The future of testing is bleak.

> biography

Peter Russell
is the Quality Development Manager at InterSystems (TrakCare) in Sydney, Australia. He studied Computing and Pure Mathematics at the University of Sydney and has a Masters Degree in Business Administration from the School of Management and Public Policy. He has an ITIL Foundation Certificate and is a CSTP (Certified Software Test Professional). Peter spent the first 10 years of his career building and managing software and the last 19 years testing and auditing it. He has deployed several large scale automated regression test environments and focuses on aligning methodologies and tools in all phases of software delivery. He was a speaker at the Mercury Interactive World Wide User Conference in 1996 where he described automation methods for Multi-platform Regression Testing. He has also published in the Journal of The Australian Computer Society.

XING We Are Hiring!

THE PROFESSIONAL NETWORK

Scrum and Kanban as it should be
Top-notch technology
Fast iterations, frequent releases

420 employees, 25 nationalities
The best colleagues you can imagine
Headquartered in the center of Hamburg

Boost your career by shaping one of Europe's foremost web applications!

XING is the social network for business professionals. More than 11 million members worldwide use XING to advance their business, job, and career.

careers.xing.com

All About Quality

by Leo Smits

Software testing is all about quality. As Kaner states [1]: 'Software testing is an empirical technical investigation conducted to provide stakeholders with information about the quality of the product or service under test.' Stakeholders need to have insight in quality, software testers provide the required information for this. The essential questions then are: What is quality of a product or service under test exactly? How can it be determined? What information about quality is exactly needed?

Quality and Decisions

There are many different definitions of quality. Garvin distinguishes five distinct approaches to the definition of quality [2]: the transcendent approach, the product-based approach, the user-based approach, the manufacturing-based approach and the value-based approach. Each approach differs significantly from the other approaches and this is reflected in the various definitions of quality. All approaches offer valuable definitions of quality for software testers.

Here however, a different perspective is presented. This perspective centers on 'decision'. The definition of quality is derived from the role it has in the context of decisions. This leads to a definition that significantly differs from the definitions of the other five approaches. On the other hand, there are common concepts. Especially the user-based approach does overlap.

First take a look at 'decision'. Individuals make decisions constantly. In many cases the decision is as trivial as 'Do I want to drink coffee or tea?' and 'What shall I take home for dinner from the supermarket?'. Sometimes individuals are faced with more difficult decisions like 'Do I purchase this car, or not?'. The types of decisions and their complexity may differ, they all basically share the same activities: identification of the available options; ascribing values to all options; order the options by value and pick the best option (the one with the highest value). An individual does not make perfect decisions. He possibly does not identify all options. Also, the values he ascribes to the various options are based on his capability to do so. But he will carry out these activities. (Note that not making a decision implicitly equals a decision.) Otherwise he cannot act.

Hence it can be concluded for decisions that:

- options must be identified;
- values must be ascribed to the options.

Value is a term that will be discussed in more detail later, but for now it is important to highlight two aspects:

- Value is subjective. It is the value as perceived by the individual. Individual A may ascribe a different value to an option than individual B. This matches the user-based approach which has the premise that quality 'lies in the eye of the beholder' [2].
- Value depends on the context of the decision to be made. Example: An individual prefers the taste of coffee to the taste of tea, and at the same time he does not want his sleep to be disturbed by too much caffeine. He will then ascribe a higher value to coffee than to tea at 9:00 AM but he will do the other way round at 9:00 PM.

What does this mean for 'quality'? From the perspective of decision making, quality has no other meaning than the value of an option as compared to the values of other options. And like 'value' to which quality is related it depends on the individual and the context of the decision to be made. There is no absolute quality in an object, idea, etc. itself. So quality can only be defined as a comparative term.

The 'comparative' aspect of quality from another perspective: Imagine a person looking at a painting. This person has never seen a painting before in his whole life, or in other words: This is the first painting he has ever seen and thus the only painting he knows. When asked, what quality would he ascribe to that particular painting? How many 'stars' would he give the painting? The fact is he cannot reasonably answer this question about quality since he knows no other paintings to compare to.

Summarized, from the decision-perspective quality contains three characteristics:

1. Quality is the value of an option as compared to the values of other options
2. Quality is based on perceived values, differing per individual

3. Quality is determined in the context of the decision to be made

An object, idea, etc. does only contain quality in the context of a decision to be made (i.e. it has no absolute quality). Within that context the object, idea, etc. is an option. It is therefore more correct to speak of 'the quality of an option' than 'the quality of an object'. The definition for quality from a decision-perspective can consequently be assembled as follows. Quality of an option is the value of that option in comparison to the values of other options, as perceived by a person and within the context of a decision to be made.

Quality and Software Testing

If quality of an option only exists in comparison to other items then what does that mean for software testing? What is the 'option' in the context of software testing and what are the other options it must be compared to?

To answer these questions it must first be clear what decisions have to be made. Let's assume that the information about quality that must be provided is intended to support a release decision for a system under test (SUT). A release decision in general has two basic options: 'Release the SUT' or 'Do not release the SUT'. There may also be other options, e.g. 'Release the SUT except for subsystem A'. Looking at the options for a release decision, options in this case are scenarios. The choice is of the kind 'What to do next?' Therefore in the context of software testing the word 'scenario' is used for 'option'.

The two (or more) scenarios can be compared. Scenarios are usable in the context of the decision to be made. The conclusion is that software testers need to identify the different scenarios for their project and then determine the value of all identified scenarios. In doing so, quality is determined in the context of a release decision and quality is determined as a comparative term.

It still leaves the question what 'value' is unanswered. It also raises the question how value can be calculated so that it covers the perception of a group of stakeholders. This challenge is caused by the fact that quality is based on perceived values of individuals whereas the release decision is not made by one individual.

Determining the Value of an Option

The first step in making a decision is determining the values of all options (considering the identification of all options as completed). What unit of measure can be used for 'value' to make the comparison in the following step possible?

Let's first look at an example: A couple goes into town to have dinner in a restaurant. They have all evening to spend and this dinner itself is the reason for their visit to the town. There are two restaurants available, one expensive restaurant with a highly acclaimed cuisine, the other being a pizzeria with affordable cooking. For the purpose of their visit the couple will decide by asking themselves 'Which restaurant serves the best food?' not 'Which restaurant is the most inexpensive?'. So the expensive restaurant has the highest quality. A week later they visit the same town again. This time to go to the movies. They have one hour left before the movie starts and still need something to eat. In this situation the couple decides by asking 'Which restaurant is the most inexpensive?'. This time the pizzeria has the highest quality. The 'highest quality' more exactly: Highest quality for them, and in the context of the current decision to make.

This example shows that quality depends on the decision to be made and that subsequently 'value' must be related to that decision as well. In the example 'value' is simply based on one simple question. In practice decision making is often more complex, but it boils down to the same idea: The unit of measurement being used supports the decision to be made.

Values in Software Testing

What unit of measurement supports the decision making for software products? What unit of measurement makes the best comparison for the scenarios?

To support the decision making process it is required that the type of value is accepted by all people involved and that it allows for a complete and still feasible comparison. This is possible when value is calculated in terms of financial value. Financial value eliminates the perceived value to a great extent. This is important because a release decision is not made by an individual but by a group. It is also in most cases relatively easy to express various aspects as a financial value. And even for difficult aspects like damage to reputation it is still possible to do so.

Looking a bit more in detail, the financial value of a scenario consists of two parts. First the SUT has benefits for the organization. Second it has costs that diminish the financial value: defects and unmitigated risks. The total value of the scenario is simply the subtraction of these two (value = benefits – costs). Neglecting the benefits (i.e. only reporting on defects and unmitigated risks) leads to an incomplete value, for example: If a choice is offered between two cars for the same price, one without damage and one with some exterior damage, then what would be the best pick? And what if the first car is a cheap small car that has run 200,000 km and the second car is a luxury car that has run only 50,000 km?

Conclusions: In software testing the value can be expressed as a financial value. The unit of measurement would be for example Euro. Also, it is important to look at both costs and benefits of the scenarios. Remember that quality only exists as a comparison of scenarios and that therefore the values of all scenarios must be calculated.

Figure 1 shows the values that must be calculated when software testing provides information about quality. It also shows that value consists of both benefits and costs. If information about quality is solely based on defects and unmitigated risks in the system under test (SUT), then only partial information is given. Also if quality is based only on the SUT and not on all scenarios, then partial information is given. This is shown as the yellow text box in the figure.

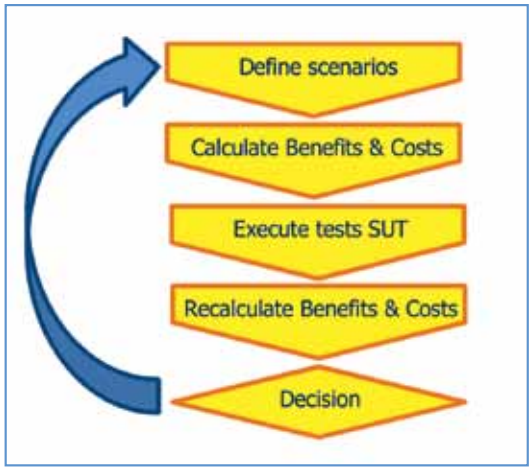
Implementation

The approach from the perspective of decision has its implications on the work of software testers. It requires additional activi-

ties from the software testing team, in particular:

- Identification of scenarios
- Calculation of values for all scenarios

The easiest way to start with identification of scenarios is to take only two scenarios to start with. It is always possible to add other scenarios later. Calculation of the values for the scenarios can be done as soon as the scenarios have been identified, before test execution starts. It provides insight in the values at an early stage, even if this information is not fully complete. When all tests have been executed, the costs for the SUT are known and the calculations must be updated. At that point the software testing team is able to provide the information about quality to the stakeholders. They can make a decision, i.e. they can choose from the scenarios. If the SUT is released, then the process ends. Otherwise the whole sequence starts all over. Two remarks can be made here. First, it might be desirable to add or remove scenarios when going into a next iteration. Second, based on the values of the scenarios it is possible to consider which defects should be fixed based on the extra value this will deliver. Note that the 'Release the SUT' scenario in principle only needs to have a higher value than the other scenario(s). There is no need to maximize the value of 'Release the SUT'. Figure 2 shows the suggested process as described here.



Some suggestions that may aid the implementation:

- Much information on value may already be available. It provides an easy starting point.
- Do not calculate values in a very detailed manner from the start. First gain experience with the new process and build metrics to refine calculations later.
- Use standard costs for defects with low severity (e.g. 'Inconvenient' and 'Cosmetic').
- Keep in mind that costs of a scenario are those costs that are made when the SUT is running in a production environment. It does not include the costs of the project to develop and test the SUT.

Conclusion

A definition of quality from the perspective of decision fits the goal of software testing well.

Following the implications of the new quality definition it appears that software testers must broaden the scope of their work. It will make work more complicated to a certain extent, but it results in providing complete information about quality. This in turn will allow decisions to be made more accurately. If software testers do not provide the complete information (i.e. leaving out benefits and scenarios), others will. That is not what should happen: Providing information about quality is the domain of software testers.

References

[1] Cem Kaner, J.D., Ph.D.: Exploratory Testing - Keynote at QAI, November 17, 2006

[2] Garvin, David A.: What Does "Product Quality" Really Mean? in Sloan Management Review Volume 26 Number 1, Fall 1984

> biography

Leo Smits
has 10 years of experience in software testing and 15 years in ICT. He has worked both as a software tester and as a test manager. As a test manager he has led the testing for several major ICT projects. Apart from software testing he shows an interest in information analysis, requirements engineering and the Software Development Life Cycle. Leo is currently working as a test manager for a Dutch insurance company. He is the owner of Smits Haltna and lives in the Netherlands.

EUROPE 2012

Testing & Finance

May 16–17, 2012

London Marriott Hotel Grosvenor Square

The Conference for Testing & Finance Professionals

www.testingfinance.com

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

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

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

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

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

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

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

[1] 25-minute presentation

Please note that the program is subject to change.

Exhibitors & Supporters

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

Testing & Finance Europe 2012 –
A Díaz & Hilterscheid Conference

Díaz & Hilterscheid Unternehmensberatung GmbH
Kurfürstendamm 179
10707 Berlin
Germany

Phone: +49 (0)30 74 76 28-0
Fax: +49 (0)30 74 76 28-99

info@testingfinance.com
www.testingfinance.com

www.xing.com/net/testingfinance

Load Test Your Web Application – Before Your Customers Do

by Mark Eshelby

Introduction: Poor Web Performance Impacts the Bottom Line

In this day and age, the web tends to be the number one way businesses interact with their customers and prospects. Web performance – or the speed and reliability of web applications – can have a huge impact on your business. According to the Aberdeen Group, a single-second delay in website response time translates to an 11 percent reduction in the number of pages customers and prospects view on your site, a 16 percent drop in customer satisfaction, and a seven percent decline in conversions.[1]

Observations from a wide variety of companies and geographies show a clear correlation between page load times and page abandonment. A page load time of just six seconds results in a page abandonment rate of about 20 percent. If the web page or application in question is a shopping cart checkout, then that business will be losing 20 percent of its customers – and its revenue – simply because of the load time.

What Can Software Testers and Quality Assurance Professionals Do?

As a software tester or quality assurance professional, your job is to ensure superior web application performance, particularly during peak traffic periods when your business often has the most to gain (or lose). If your web applications are slow, you're going to lose customers to the competition. If customers can't get through their transactions, you're not going to make money.

Years ago, load tests were designed to validate that your database, server, and internal network were ready to handle peak traffic loads (in the vicinity of one hundred users!). As web applications have grown significantly more complex over the past decade, they now require a whole new set of techniques and practices for load testing.

Load Testing from the End User Perspective

Today's web applications have evolved from simple, single-function tools to feature-rich composites offering extensive functionality from external third-parties like shopping carts, ratings and reviews and analytics, as well as RSS feeds and social media plug-ins. In fact, the average website today connects to more than eight hosts. While these extensive functionalities can enable a richer online customer experience, they can also introduce performance risks since any one component can cause a web application to slow down. Case in point: Tagman[2] recently determined that Google's +1 plug-in drags down page load times by a full second.

In addition to third-party applications, there are many other performance-impacting elements standing between your data center and your end users around the world, including cloud service providers, regional ISPs, local ISPs, content delivery networks and even browsers and devices. Collectively, this concept is known as the web application delivery chain, and poor performance anywhere in the chain will degrade the cumulative end user experience and reflect poorly on you, the web application owner, regardless of the actual cause.

The reality of modern web applications is that even if your tools inside-the-firewall indicate that everything is running OK, that's no guarantee your end users are happy. You can no longer just test inside the firewall, because this gives you only partial coverage and leaves you with significant blind spots. Many aspects of the end-user experience will not be inferable from data collection points within the data center itself. The point at which the end user accesses a composite application is the only place where true application performance can be assessed.

Today, it becomes more critical to assess the end-user experience as part of an overall load testing strategy canvassing all performance-impacting elements – from the end user's browser, all the way back to the multiple tiers of the data center and everything in between. This is the key to identifying and fixing any weak links. Some businesses believe they can't load test external elements because those elements are beyond their control. However, if you can do a load test from the end user's point of view and include that entire web application delivery chain in your load test, then you really are in a stronger position of control.

1 *Aberdeen Group, "The Performance of Web Applications: Customers are Won or Lost in One Second."*

2 *http://blog.tagman.com/2011/07/how-google-could-cost-online-retailers-millions-2/*

Real World Examples

Software testers and QA professionals can design tests that combine load generated from the cloud with load generated from real end-user desktops and devices, in order to gauge how performance for real end users fares under heavy traffic loads. Once you detect that a particular end-user segment is experiencing a performance degradation, advanced mechanisms can then accurately and precisely pinpoint the root cause, whether it's inside or outside your data center. This is critical because most business organizations don't have time to test repeatedly to try and fix slow, poorly-performing web applications during load tests. They need to get to the root cause of performance delays in minutes, not days or weeks, with the goal of fixing performance problems in the fastest, most efficient manner possible.

Existing solutions on the market may be able to tell you the health of a server during a load test, but you must go deeper than that – as deep as identifying the offending method, API or database call at the root of a performance problem. Gaining a 360 degree view of all the elements impacting the end-user experience under load is the only way to ensure fast, accurate problem identification and resolution, ensuring that performance issues under load are fixed before, during and after an application goes live.

For example, let's say you determine that performance for a particular key end-user geography slows under heavy load, due to a regional ISP. You can now make one of several informed choices: you can lighten your site content which may help enhance speed; or, you can strategically enlist the services of a content delivery network to get content "closer" to these end users and enable faster downloads.

Conclusion: Don't Wait

Whatever you do, don't conduct your load testing from the end-user perspective on Monday, if your application is going live the following Tuesday. Load testing from the end-user perspective often yields many opportunities for optimization along the complete web application delivery chain and you need time to be able to intelligently exploit these. That's why it's also important to commit to load testing not just before, but during and after application production, as opportunities for optimization will continue to present. Ultimately, load testing from the end-user perspective results in a more strategic, more informed test that will help guide decisive action addressing performance-impacting elements along the complete web application delivery chain.

> biography

Mark Eshelby

As a senior product manager at Compuware, Mark Eshelby is responsible for developing the strategy for the Compuware Gomez load and performance testing offering. Mark has worked in the automated software quality field for 20 years in product management and consultative roles. Mark's experience with load and performance testing dates back to the mid-90s and he has worked with hundreds of companies around the globe on their performance testing objectives.

ISTQB Expert Level: Test Management for Complex Systems

by Dr. Armin Metzger, Jörn Münzel, Dr. Frank Simon & Dr. Stephan Weißleder

Complex systems require competence in Quality Assurance and Testing

Today's IT projects are getting more and more challenging in terms of required quality, possible time and available budget. Typical characteristics of their implementation are globally dispersed teams, many different programming paradigms and techniques used, endless lists of peripheral systems that have to be integrated, consideration of different and partially national regulations, and the necessity to synchronize different processes and release plans. For the people involved in the development of such complex systems the challenge is not restricted to planning, design, and implementation. In particular, there is a need for skills enabling scalable and continuous quality assurance and testing. The corresponding methods have to be carefully applied to guarantee a well-defined total system quality. This means that complex systems require a high competence of the people involved in quality assurance and testing.

Test managers must smoothly integrate testing in the overall process

From a quality assurance point of view, two system levels are important: The first one is focused on single modules and applications being part of the overall system. The second one considers the end-to-end-quality, based on the quality of the first level, but enhancing it by additional aspects. It's the test manager's task to ensure that every relevant module/application has a corresponding quality and that these different quality assurance activities are synchronized for enabling assuring and testing the end-to-end-quality. In the end he is responsible for the total quality that is delivered. Additionally he is responsible for the alignment between achieved quality and desired quality (not necessarily perfect quality) by using existing testers and corresponding methods and techniques.

The importance of his role is often undermined by high time pressure and the fact that testing is the last step in the overall process chain in many organizations. This balancing requires exquisite skills for the test manager. On the hard skills side, this does not only mean detailed knowledge about different testing techniques and their related advantages and disadvantages, but includes orchestrating them to an overall testing strategy and synchronizing them into an overall test plan. On the soft skills side, the increasing complexity of today's IT projects requires more and more management skills, consisting of leadership capabilities, stakeholder management as well as the ability to smoothly integrate own processes into an overall process chain.

Typical examples are

- controlling single testers and their tasks
- synchronizing test activities between different test factories dispersed worldwide
- efficient usage of test environments and test tools
- aggregating test reports to an overall quality report
- adjustment of test plans after a change in the project master plan

These challenges are even more difficult to fulfill in an agile environment, where many fine granular iterations each require a detailed test report to be used as input for the next iteration. Maintaining long-term plans with a reservation scheme for necessary test environments and peripheral systems is a challenge on its own.

Where to learn those skills to be a suitable test manager? How to get familiar with test challenges in complex systems? What certifications exist today as indicator for knowing the test Best Practices? How much money do we spend in the competence of our own or others' skills to be successful test managers?

Become a Certified QA and Test Manager

In this section, we present the results of two studies and the corresponding offers of the ISTQB and its German national board, the GTB.

The software test study [C] contains the results of an online survey with the topic „software test in the practice" from the year 2011. All in all, 1623 people from small to big companies from German speaking countries (DACH) participated in the survey. More than 50% of the participants completed the questionnaire. The goal of the survey was to investigate if the state of the practice from 1997 until today has changed.

The study contains five statements about the expected changes from 1997 to 2011. In the following, we describe the main results: Today, quality-assuring means are more often applied in early phases than in 1997 - the focus of quality assurance remains on the late phases. There are, however, differences in safety-critical domains. Furthermore, agile development approaches are used by 25% of all participants, which is an increase compared to 1997. Scrum is the main player in this field. Despite the high expectations, however, agile engineering processes do not result in higher quality. For instance, domain experts are included in only 33% of the agile projects - in contrast to 50% in classic phase-oriented projects. Today, the job of a tester has a much better reputation than 14 years ago. 77% of the participants stated that tests are run and managed by trained testers. 74% of testers and managers are aware of the ISTQB training scheme. 70% of the participants are certified Foundation Level testers - 90% of them consider this training helpful. The survey also shows that the planned Expert Level modules meet the actual interests of the participants. Furthermore, test automation has gained a lot of attention: more than 26% of the participants stated that their unit tests are 100% automatically executed. Here, too, the agile approaches with only 43% automated tests show some weakness. There seems to be no trend towards the outsourcing of unchallenging testing activities. Contrary to the expectations, the primary goal of quality assurance is the increase of the product quality; decreased costs are ranked only second place.

The BITKOM study [D] is focused on the chances of training in the sense of a life-long learning and the corresponding support of human resource departments of companies in the telecommunication domain. The opportunities that trainings give are important for the struggle against an insufficient number of domain experts. Companies that offer training to their employees are very attractive for experts, especially for the younger ones. Companies in the telecommunication domain offer 4.5 days of training per year, which is considerably higher than the inter-domain average of 2.5 days per year. The study also shows that companies are more successful if the management steers the personal qualification programs compared to companies in which employees manage their training. Certificates like, e.g., the ones of the ISTQB, are a strong motivation for personal qualification. Companies apply all kinds of trainings from in-house seminars to web-based seminars.

An extensive and state of the art, high-quality training is a mandatory prerequisite and often a key factor for the success of products and projects. Hard skills and soft skills of test teams are both important and should be checked already during recruiting. In this respect, the worldwide known and leading qualification standard of the certified tester scheme has proven itself. The International Software Testing Qualifications Board (ISTQB®) is responsible for this scheme, which in Germany is supported by the German Testing Board (GTB). The tester certification schemes „Foundation Level" and „Advanced Level" cover general quality assurance techniques as well as testing independent of programming practices or paradigms. Both schemes are well established with more than 20,000 certificates in Germany alone (as of 2011-09-30).

ISTQB Expert Level: The „black belt" for testing complex projects

A few months ago, the ISTQB introduced two syllabi for a new, third level of the Certified Tester qualification, called "Expert Level", which adds to the Foundation and Advanced Level. This Certified Tester Expert Level will be continuously upgraded and extended in the future by further syllabi and testing related topics. The new level targets experienced testing professionals who want to deepen their know-how and specialize in certain fields of test management or testing. The "Expert Level" courses are bundled in modules which can be selected separately. The syllabi for two of these modules are already finished and have been released: "Test Management" and "Improvement of the Test Process" both concentrate on the management aspect of testing – especially of testing for large-scale, complex and heterogeneous IT systems. The more "technical" modules on "Test Automation" and "Security Testing" are in development.

Certified "Experts" will be able to improve both effectiveness and efficiency of the overall testing workflow as well as the test suites themselves. Here, "testing" is not only considered within the scope of software, but within the scope of systems: "[A] system may also include hardware, middleware and firmware." ([A], p. 14). With such a comprehensive concept of testing at its core, the "Expert Level" has to cover a wide spectrum of know-how.

In the "Test Management" module experts get to know the strategic aspects of testing and testing workflows: the influence of businesses' strategic and quality objectives and their influence on testing methodology, standard testing strategies, various ways to measure and control the effectiveness, efficiency of tests etc. They also learn how to align testing to business strategies in order to improve the impact on desired business outcomes.

In the "Improving the test process" module, experts learn how to improve existing test processes and tests. Typical reasons for improvements are explained and the model-based and analytical approaches are introduced, which can be taken for the creation and the implementation of improvement programs. A thorough examination of generic improvement approaches, such as the "IDEAL framework" as well as in-depth coverage of model-based and analytical approaches to test process improvement deepens the experts' methodological knowledge. Chapters on management, leadership and change management enhance their social skills.

Further information about the Expert Level is available from the ISTQB website as a downloadable overview document [B].

Other training courses

In addition, the German Testing Board offers another GTB-developed training scheme for experienced testing professionals. This scheme allows for learning or refreshing technical and soft skills which are vital for the work as software tester.

This scheme, the "TTCN-3 Certificate", has been very successful so far. TTCN-3, or "Testing and Test Control Notation Version 3", is an internationally standardized language for testing and certifying which was established by ETSI. The test-oriented and implementation independent semantics of the language supports object-oriented programming, encapsulation, modularization and re-use, which makes it a means to both improve test design efficiency and ensure the re-usability of test automation and certification suites. The language was established e.g. within the telecommunications industry, but has started to play a bigger role in other industries as well. Currently, the TTCN-3 Certificate® is only offered by the GTB in Germany.

Summary

The ever increasing requirements in functionality, performance and quality have to be fulfilled by large and complex IT systems. This demands, beside the knowledge about IT architectures and suitable programming techniques, also appropriate methods, processes and organization in quality management, assurance and testing.

We need stability and flexibility; stability within the fundamentals of our knowledge, our education and our basic roles, and flexibility and openness in the approach we apply in projects. Both sides are especially important for the area of quality management, assurance and testing in large complex systems with heterogeneous development and dispersed teams. We need experts and specialists in testing. The ISTQB certification scheme is a powerful and proven collection of best practices for those experts. The new ISTQB Expert Level is the ultimate level for those test managers who want to demonstrate their being an expert and who are willing to maintain this level of expertise.

References

[A] ISTQB(R) (Eds.): Certified Tester Expert Level Syllabus. Improving the Test Process. n.l., 2010

[B] Click downloads at http://www.istqb.org

[C] A. Spillner, K. Vosseberg, M. Winter, P. Haberl: "Softwaretest-Umfrage 2011", available at www.softwretest-umfrage.de

[D] Bitkom Servicegesellschaft mbH, Kienbaum Management Consulting, F.A.Z.-Institute: „BITKOM-Studie: Weiterbildung in der ITK-Branche 2011", available at http://www.bitkom-service.de/files/documents/Weiterbildung-in-der-ITK-Branche-2011.pdf

> biography

Dr. Armin Metzger
is department manager at sepp.med GmbH. He is responsible for IT multi-projects in complex and safety relevant domains. He has almost 20 years of experience with quality assurance, development and processes in scientific and industrial projects. Dr. Armin Metzger is a founding member of the GTB and member of the GTB board of directors.

Dr. Frank Simon
is head of SQS Research at SQS Software Quality Systems AG. He investigates and analyzes current challenges and opportunities in IT projects to develop new services around quality management and testing for a holistic quality service orchestra.

Jörn Münzel
is head of the Competence Center QA and Test at ITinera Consulting. He has over 25 years of experience in software development, quality assurance and testing. His current focus is in improvement of test effectiveness in finance areas, especially in stress test simulation.

Dr. Stephan Weißleder
works as the Research Manager Testing at the Fraunhofer-Institute FIRST with focus on test automation and model-based testing. He is also a team member of the model-based testing community. His goal is to push the exchange of industrial experience and academic theory.

All authors are members of the German Testing Board e.V. (GTB) and can be contacted by <firstname>.<lastname>@german-testing-board.info.

Agile TESTING DAYS

November 19–22, 2012
Potsdam (near Berlin), Germany

Call for Papers at www.agiletestingdays.com

Focus on testing: Radical market changes put pressure on quality assurance teams

by Murat Aksu & Stefan Gerstner

World Quality Report 2011-2012: Testing estimation methods and test automation tools remain underutilized

The recent economic downturn forced businesses around the world to run their IT operations with smaller budgets and fewer resources. Without much money to spend on new application development, IT managers had to make the best use of their existing systems, ensuring that they function without fail to support the companies' core business processes. This lean approach has increased the pressure on quality assurance (QA) teams to put in place solid processes, methodologies, and tools for thorough validation and monitoring of applications' functionality, availability, and performance. As the economy continues to recover globally, the focus on quality remains vigilant. Most companies are not willing to go back to the old practices of supporting a sprawling landscape of outdated and disjointed legacy systems and outdated technologies. Application modernization and consolidation initiatives are on the rise across all industries, and QA is playing a critical role in ensuring that this process goes smoothly. These trends are confirmed by the findings of the respondents of the 2011-2012 World Quality Report.

QA budgets remain under pressure

The new trends, technologies and initiatives are increasing the workload for QA teams. However, there's no exponential corresponding increase in QA budgets to support the added pressure. Only five percent of companies have significantly amplified their QA budgets, and surprisingly 13 percent still don't have dedicated QA budgets at all. Just over a third of respondents indicate that their quality-related budgets have increased slightly, and an additional 35 percent suggest that they stayed the same (see figure 1).

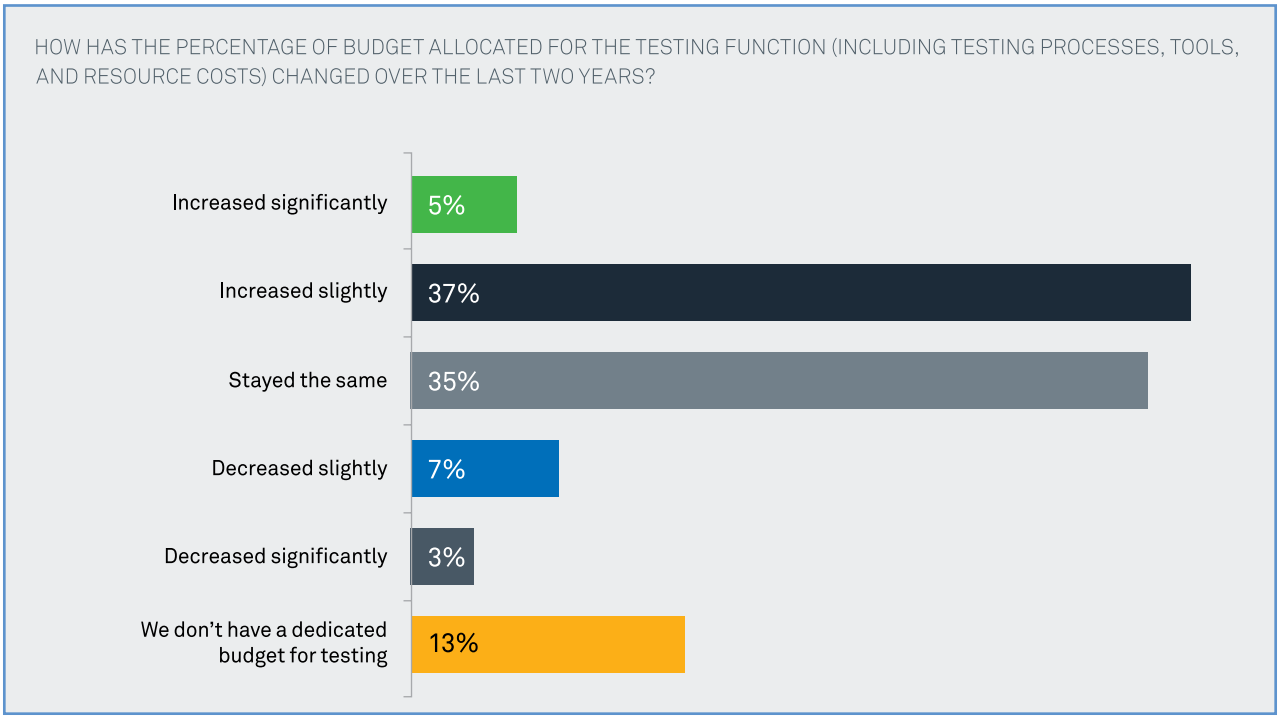


Figure 1: How has the percentage of budget allocated for the testing function (including testing processes, tools and resource costs) changed over the last two years?

Even though QA teams are making an effort to tie their results to the business goals and quantify the losses from production defects, there still aren't any established standards for measuring QA success or estimating the amount of time and resources needed to sufficiently test an application. Only 22 percent of respondents say that they use industry standard estimation methods, with an additional 37 percent indicating that they use internally developed estimation techniques, and further 30 percent admitting to approximate the effort required for application validation based on past experiences or as a portion of the total development effort.

Large companies and developing countries welcome automated testing

For the third consecutive year we've observed the rise in the use of test automation technologies; however, manual testing is still often the preferred method for application verification. Not surprisingly, the degree of test automation varies by company size and geography (see figure 2).

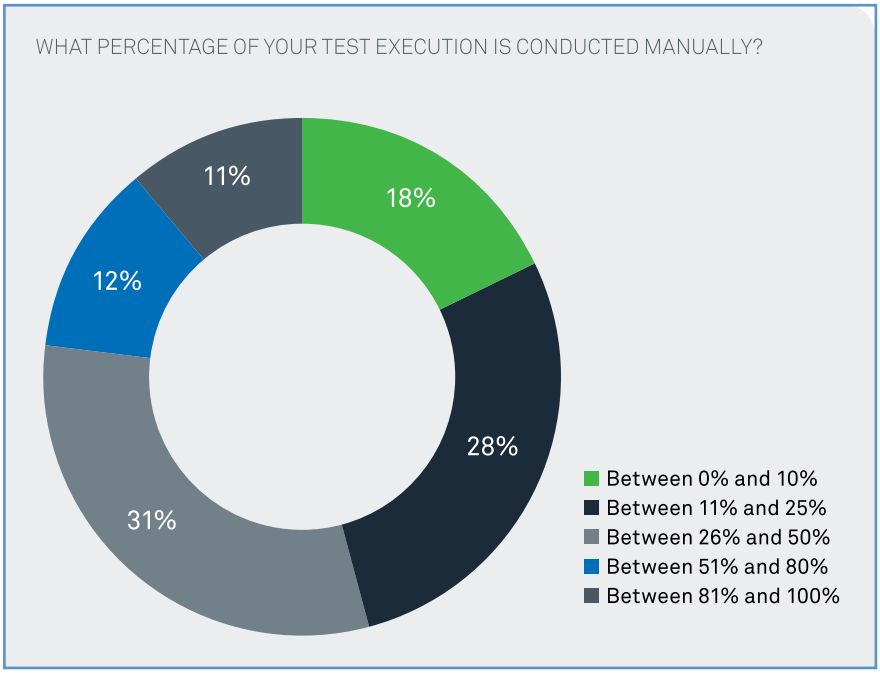


Figure 2: What percentage of your test execution is conducted manually?

For example, a quarter of small company respondents indicate that nearly all of their tests are run manually. In contrast, only about five percent of medium and large enterprise respondents say that close to 100 percent of their tests are manual. China appears to be in the forefront of automation use with only six percent of respondents suggesting that more than half of all tests are being run manually, followed by Western Europe and North America. This can be explained by the fact that often developing countries have the ability to leapfrog ahead of the developed world and use the latest technology available. North America appears to be lagging behind in test automation with an average of 28 percent of American and Canadian respondents suggesting that more than half of their tests are being run manually. In comparison, only 14 percent of Western European survey participants say that most tests are executed manually.

German companies in particular are rather more advanced in their use of automated testing. An overwhelming majority (90 percent) of respondents use automated test execution for more than half of their work. This is 13 percent higher than the international average. It appears that the technical preconditions for a more industrialized approach to testing are in place, although automation is not yet being used to the full extent, due to a limited influence of the QA role within German organizations. Despite the lower automation numbers, we observe an increase in the use of automated testing methods and tools. Business pressures for faster time to market and reduced costs can explain part of this trend. The rest of the picture is formed by the efforts of testing software vendors to develop increasingly easy-to-use automation tools that require minimum scripting and development effort.

Outsourcing is increasing across a broader number of geographical bases

For the third consecutive year, we have seen outsourcing in testing activities and its growing role in the quality management process. Over three-quarters of survey respondents say that their companies employ the services of contractors or third-party vendors for quality assurance.

At the beginning of the outsourcing movement, companies were seeking cheaper resources to augment their internal capabilities at lower costs. Today's providers must offer a complete suite of services – including testing strategy, requirements definition, functional and performance testing, user acceptance testing, and security testing.

The Capgemini Germany study IT Trends 2011 , reveals that today the average German IT department provides 50 percent of its own IT services, while leading companies with more industrialized IT departments have reduced the volume of services provided by their own department to 15 or 20 percent. That study also shows that German companies have a preference to outsource services to Central European or nearshore locations rather than to offshore locations such as India or China.

Compared to other countries, German companies have a lower percentage of testers working at a nearshore or offshore location outside the company's main offices. Over 40 percent of test resources are located within their own office facilities. Nearly half (47 percent) of respondents have between 19 percent and 25 percent of their testers in nearshore and/or offshore locations. Only 12 percent of respondents have between 26 percent and 50 percent of test resources abroad, and none above 76 percent.

The average of all other countries has more than double the amount of testers abroad. The reasons for these differences are partly language issues, since a majority of companies work in a German language environment and are reluctant to change this. It also reflects the fact that German companies have large IT departments and use IT service providers primarily for getting access to specialized resources. The answers regarding the ideal geographical location to contract and/or outsource testing activities also confirm these findings. Almost half (48 percent) of German respondents prefer contractors co-located with their own employees. This is nearly double that of the international average. For 11 percent of respondents, Eastern Europe is the highest-ranked outsourcing location, while India and China are rated significantly lower than the international average.

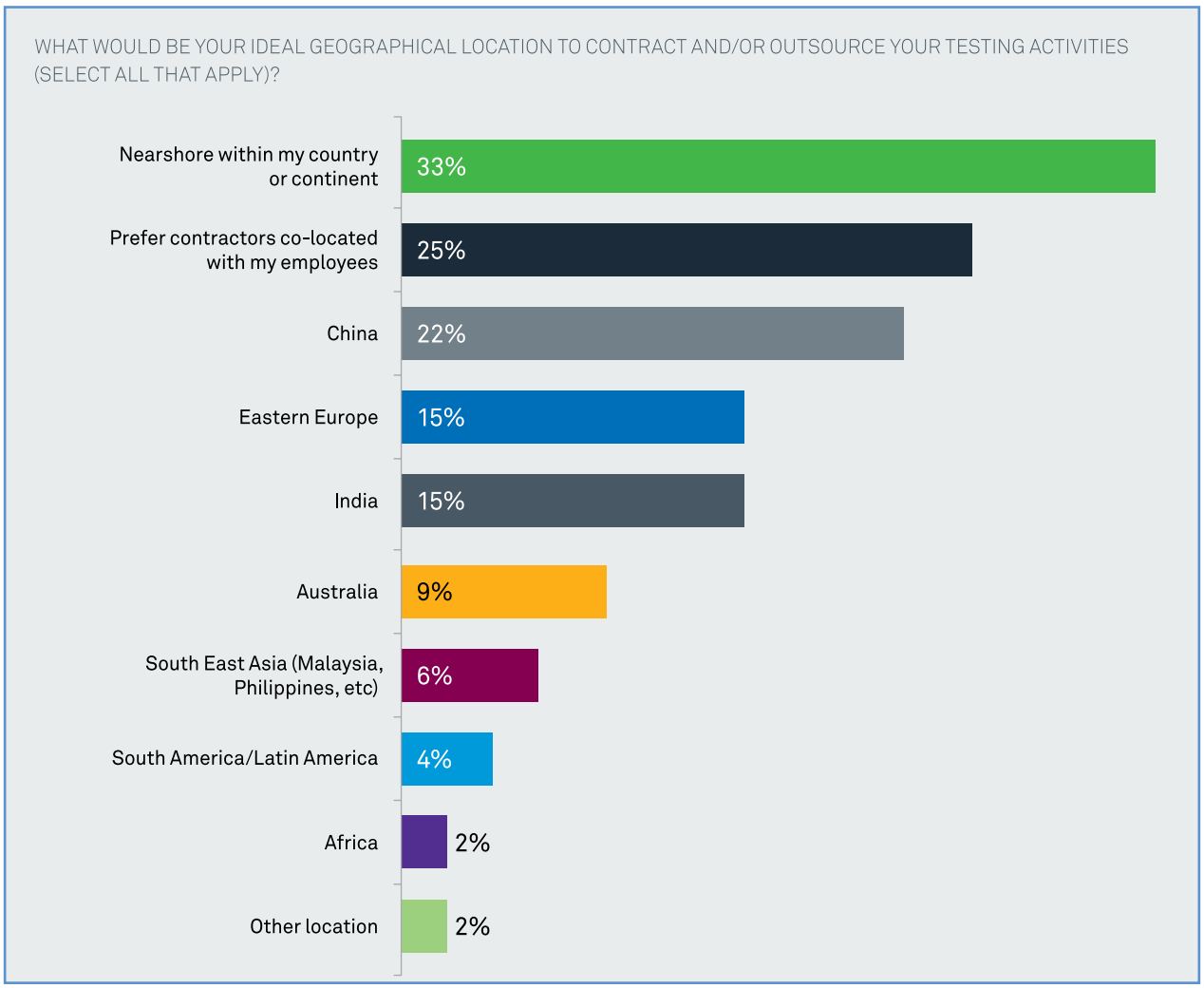


Figure 3: What would be your ideal geographical location to contract and/or outsource your testing activities? (select all that apply)

Using the cloud to reduce IT costs and increase agility

The model of deploying and consuming IT services on the cloud infrastructure is quickly gaining traction. Compared to last year's survey, the percentage of companies that are moving at least some portion of their IT systems to the cloud has grown by five percent.

The cloud has a great impact on the world of quality and testing – both for testing on the cloud and testing of the cloud itself. Using the cloud while testing offers obvious benefits – instead of installing many servers for load and performance testing, testers can use the elastic cloud to generate load on their applications. Similarly, the cloud can be used as a separate test environment or as a pay-per-use model for testing software applications. A much more complicated use case is the testing of cloud-based applications and infrastructures. If an organization's IT portfolio contains a mixture of cloud-based and internally-hosted applications, maintaining the quality of the portfolio becomes a combination of traditional QA verification and the management of contractual terms and SLAs. A QA manager, with a portfolio containing a cloud-based CRM system for example, cannot afford to test it in a vacuum without understanding how the provider's security, performance, availability, and backup procedures might affect the entire end-to-end business process. Testing cloud applications is a three-pronged approach. It still has the traditional user-experience testing element, but it also adds a service-level testing component below that, and the end-to-end business process testing above it. This approach allows testers to validate the application infrastructure in the shared model, as well as exercise the end-to-end processes that touch multiple applications.

Germans more cautious towards the cloud

Cloud is perhaps the biggest game changer in the art of testing requiring a different skillset and a different portfolio of automation tools to successfully validate the functionality and performance of complex applications that are running on a shared infrastructure. Testing in the cloud is more about managing the entire IT portfolio of services, rather than verifying the quality of individual systems.

The approach of German companies towards cloud computing seems to be more cautious compared to other countries. The vast majority of respondents (67 percent) expects that only a quarter of their applications will be hosted or migrated to the cloud over the next year, while the international majority expects to host or migrate 50 percent, or more of their applications to the cloud. This careful attitude is confirmed by the clear preference for standard office applications (mail, agenda, word processing, etc.) as typical candidates for cloud computing. Over half of German respondents favor this option compared to 33 percent for business-critical systems.

Security testing is shaping up

Application security is typically viewed as an audit and risk management function, and different companies place this responsibility with different organizations, although this distribution varies widely by geography, industry, and company size. The larger

the company, the greater the percentage of respondents who say that they have a dedicated Information Security team – a specialized group of security professionals who help design the company's security procedures and requirements. Over half of respondents from China and nearly half from Brazil say that the primary owner of application security at their companies is Information Security. In contrast, respondents from North America and Western Europe state that they rely equally on QA and Information Security, followed closely by Operations. This suggests that perhaps in more mature markets, security testing is becoming a more mainstream discipline that is being incorporated into the application lifecycle. Unfortunately, even companies that have a dedicated Information Security team don't always follow their recommendations. Our survey also finds that there is a significant lack of communication and cooperation among different teams involved in security testing.

Most commonly, organizations spend their security funds on network security. Since the introduction of the first corporate networks, companies have been aware of the potential security issues such as unauthorized access or denial-of-service attacks. Over time, firewalls were invented, standards introduced, and now, most companies automatically assume the safest settings when they build their networks. Application security on the other hand is a lot less mature, and application security testing often lacks common goals, standards, and tools. In our survey, very few respondents admit that they are engaged in automated dynamic (6 percent) or static (10 percent) security testing. Most companies perform application security audits by Information Security teams (42 percent) or have manual source code reviews (21 percent).

Security testing is an evolving discipline. Just a few years ago, it would have been acceptable to focus solely on finding vulnerabilities in existing applications and patching them to remedy any potential problems. This approach, however, can no longer keep up with the pace of innovation among the attacker and hacker communities. The only way guaranteed to ensure that applications are secure is to build security into the entire development process, including requirements, design, and code, and develop IT systems to the highest security standards from the start.

Bibliography

1. World Quality Report (WQR) 2011-2012 – global survey of more than 1,250 CEOs, CFOs, CIOs, IT directors and managers, and quality assurance directors and managers around the globe. The goal of this report is to examine the state of application quality and testing practices across different industries and geographies. Published by Capgemini, Sogeti and HP June 15, 2011. http://www.de.capgemini.com/insights/publikationen/world-quality-report-2011-2012

2. IT-Trends 2011 – a study of the changes in IT and its impact on different industries, published by Capgemini Germany February 16, 2011. www.de.capgemini.com/insights/it-trends/

> biography

Murat Aksu
is Vice President at Capgemini and has been in the IT industry for many years, amongst others as Head of Quality Assurance for the Finance Division of Capgemini. Previously he was Director of Product Marketing responsible for performance testing and SOA toolsets at Mercury Interactive (now HP Software & Solutions).

Stefan Gerstner
is Vice President Global Sogeti Services and has more than 20 years experience in IT services. For 12 years, his focus has been on business development for testing and quality assurance.

Agile testing is one of the best practices – with a proper test plan and strategy

by Hemangi Laxman Gawand

'**Testing results are an image of the developer's hard work**', this is not a quotation from a famous celebrity, but from a software professional who has seen the complete lifecycles of multiple projects. Every time we conclude a study or survey or task force on the subject of the software development process. However, there is always a need to follow '**the best practices in software industry**' to minimize cost and manpower. There has been a constant need of up-grading, not only in the 'development' cycle, but also in the 'testing' process.

This article focuses on various basic practices in the industry and also mainly on '**agile testing practices**'. Agile testing is one of the key software testing practices which completely follows the principles of agile software development. In this practice, the emphasis is not just on testing procedures. It mainly also focuses on ongoing testing until quality software from an end customer's perspective has been achieved.

1. Introduction

The separation of debugging from testing was initially introduced by Glenford J. Myers in 1979. In 1988, Dave Gelperin and William C. Hetzel classified the phases and goals in software testing in the following stages [7]:

- Until 1956 - Debugging oriented
- 1957–1978 - Demonstration oriented
- 1979–1982 - Destruction oriented
- 1983–1987 - Evaluation oriented
- 1988–2000 - Prevention oriented

In the past two decades, there was a major inclination towards the Waterfall model in software industry. Testing that can be defined as a common practice that is performed by an independent group of testers after the functionality is developed, before the

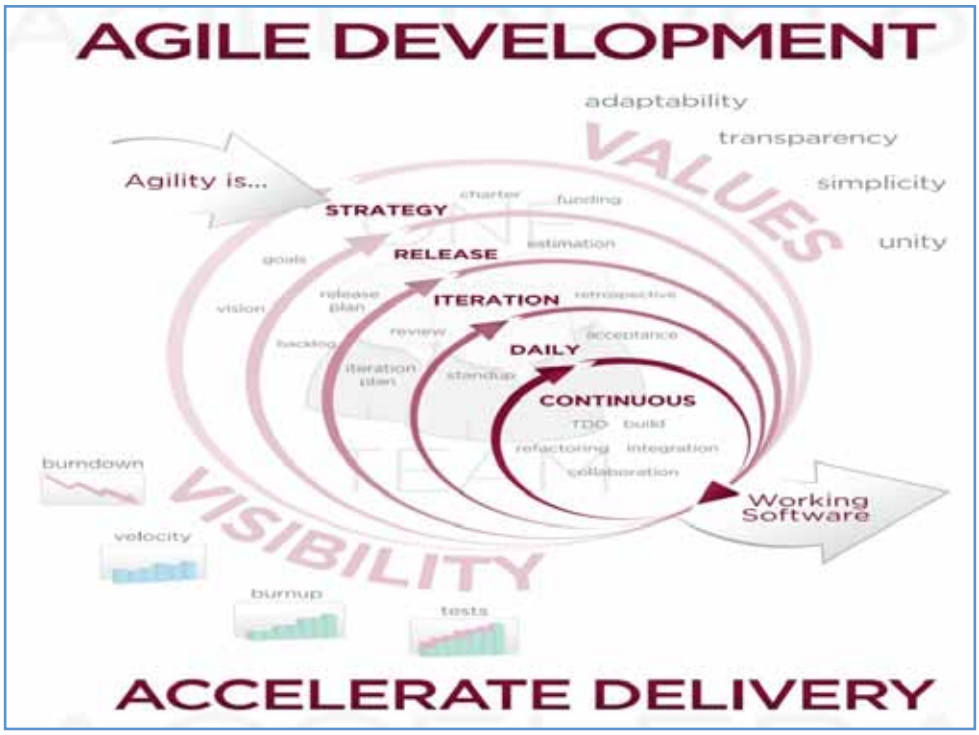


Figure 1: Agile Development Diagram

software is shipped to the customer. This practice often results in the testing phase being used as a project buffer to compensate for project delays, thereby compromising the time devoted to testing. Even though this is wrong, many major projects use the testing phase time for development work and hence cause that the testing phase suffers to a great extent.

In contrast, some emerging software disciplines, such as extreme programming and the agile software development movement, adhere to a „test-driven software development" model. In this process, unit tests are written first by the software engineers along with the development work.

2. Need for an agile methodology

Incremental software development methods have their roots back in 1957. In 1974, a paper by E. A. Edmonds introduced an adaptive software development process.

One of the similarities of the agile methodology and the traditional method is to conduct the testing of the software as it is being developed. The unit testing is conducted from the developer's perspective and the acceptance testing from the customer's perspective. The key difference is that in the agile method the customer and developers are in close communication, whereas in the traditional method, the „customer" is initially represented by the requirement and design documents. Hence, the agile method minimizes the gap between the customer, developer and tester.

Agile testing is a dynamic approach to testing. In this testing, the requirements are not stable which means that they keep changing according to customer. Figure 1 gives an overview of agile development to help understand the agile procedure.

Listed below are few of the golden principles on which Agile is based. A thorough understanding of these principles is needed prior to its implementation.

1. Customer satisfaction by rapid delivery of software is a must.
2. Always welcome changing requirements, even late in development.
3. Working software is delivered frequently mainly in two weeks.
4. Keep track of modules delivered as well as in progress state.
5. Maintain a constant pace.
6. Close, daily co-operation between business, developers and tester.
7. Face-to-face conversation with customer / requirement owner.
8. Motivated individuals are need to form a great team to work with.
9. Continuous attention to technical excellence, good design as well as constant evaluation.
10. Make life simple by keeping the design as simple as possible.
11. Self-organizing teams.
12. Regular adaptation to changing circumstances. Flexibility is the key factor.
13. Daily stand-up meeting to answer questions like: What did you do yesterday? What is today's task? Any hurdles in task achievement?

Agile testing is built upon the philosophy that testers need to adapt to rapid deployment cycles and changes in testing patterns as defined in the agile methodology.

3. Agile methodology

To truly understand agile testing and quality strategies, you must understand how they fit into the overall agile system development lifecycle (SDLC). Figure 2 depicts a high-level view of the agile development lifecycle.

This SDLC is consists of six phases:

1. Iteration -1,
2. Iteration 0/Warm Up,
3. Construction,
4. Release/End Game,
5. Production, and
6. Retirement.

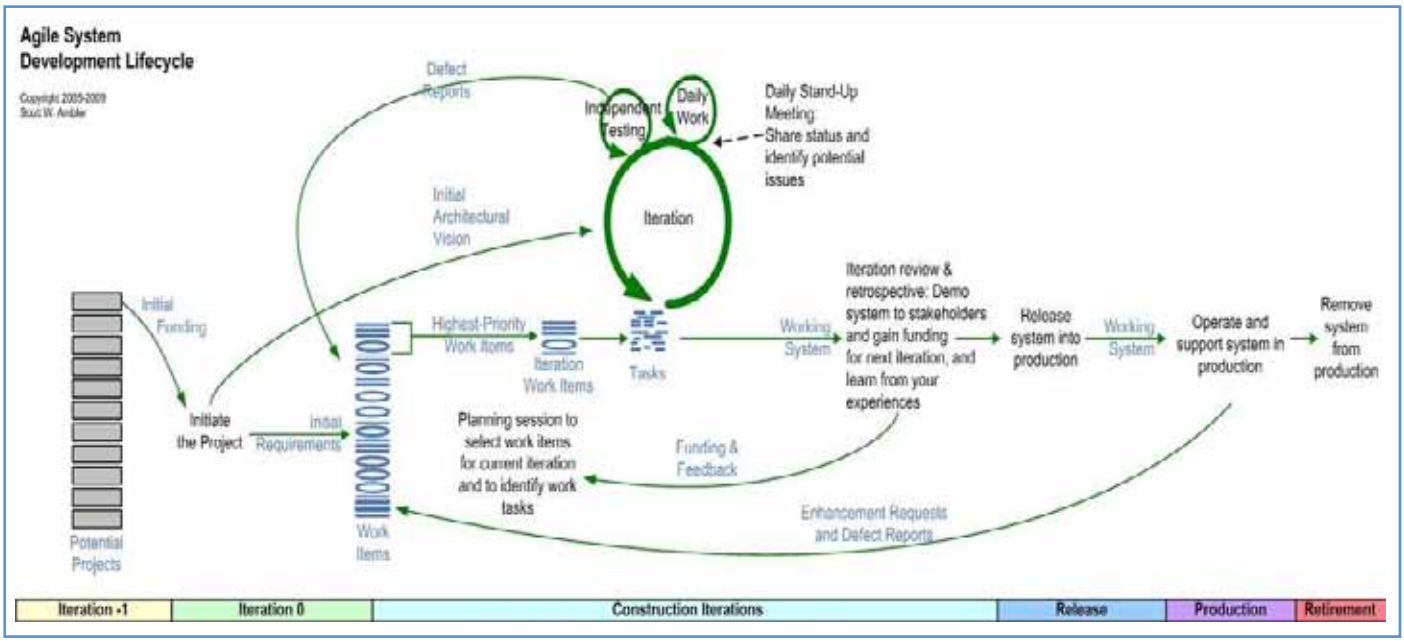


Figure 2:- A detailed agile SDLC.

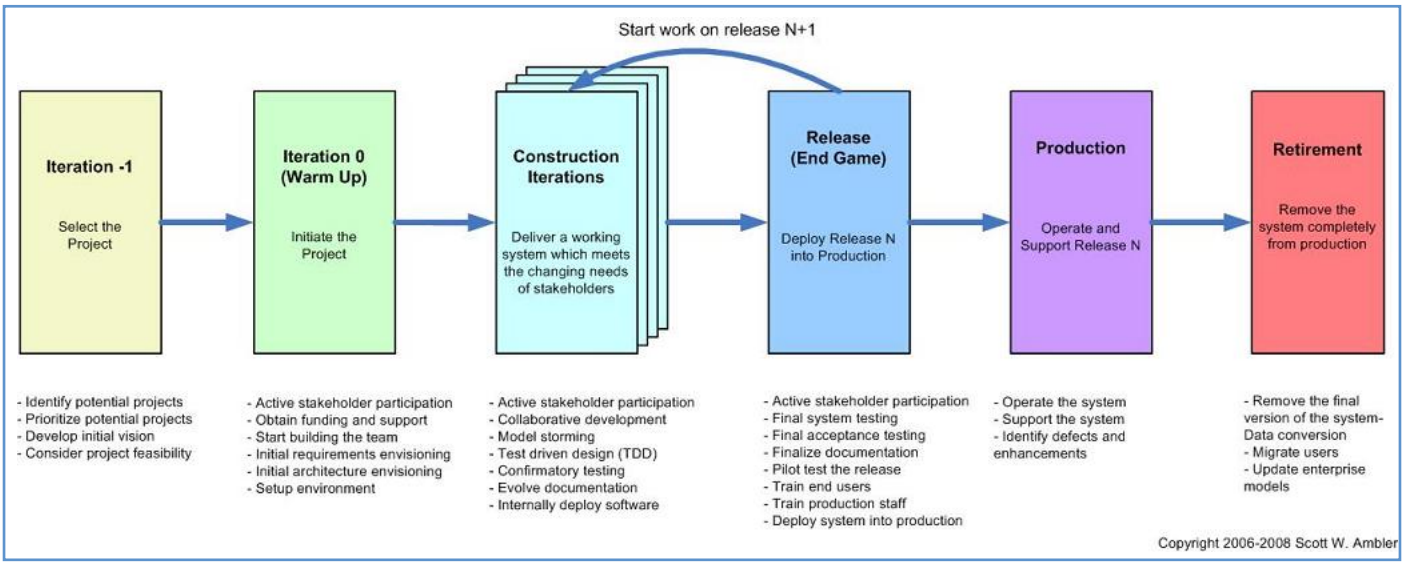


Figure 3. The Agile SDLC (high-level).

3.1 Rules of agile testing

Listed below are important rules to be followed when testing by agile methodology. You should keep this questionnaire handy when writing test cases.

1. Who is/are the customer(s) and who will be the target user group?
2. Why is the need for testing on the said system specified? Is there any special dependency on the system/s?
3. Which user/usage goals should be met in the current delivery cycle?
4. What user problems should be solved?
5. Which user benefits should be achieved?
6. Which functions and characteristics are included in testing?
7. What is supposed to happen? What sequence of actions? Do you / the team have a sound understanding of the user story?
8. Which test case is to follow which test (interdependency of test cases)?
9. What are the most common and critical parts of the functionality from the user's point of view that have to be satisfied first?
10. Are there any performance requirements included? Performance test cases must also be included in the test plan if the requirements demand this.
11. What is an acceptable response time for the users?
12. How tolerant should the system be to faulty input or user actions? Is system dependency included in test cases?
13. What are the limitations in the software and/or hardware regarding characteristics, features, functions, data, time and space? Which exceptions are assumed directly and/or indirectly? Clearly mention assumption prior test.
14. Is the description complete enough to proceed and decide how to design, implement and test the requirement, involved features and the system as a whole?
15. Which problems and risks can be associated with these requirements?

3.2 Agile testing strategy

The agile testing strategy is part of the overall test planning. Listed below are a few aspects of this strategy prior to implementation.

Define the test scope.

The testing scope has to be well defined for each of the scrum deliverables. The following questions need to be answered:

1. What needs to be included?
2. What needs to be excluded in testing for this particular project in the current release?
3. What is new in this release?
4. What has been changed or corrected for this product release?

Identify levels and types of testing needed.

1. What system integration levels are needed
   - modular
   - functional
   - subsystem/system
2. What are the test types?
   - Structural
   - Functional
   - Characteristics
   - Performance
3. What are the test objectives?
   - Acceptance test
   - Individual functional test
   - Integration test

Define test goals.

What testing needs to achieve the desired goal in the project?

Knowledge Transfer – The Trainer Excellence Guild

Díaz Hilterscheid

The Whole Team Approach to Agile Testing

by Janet Gregory

Date	Place	Price
Jan 9–11, 2012	Madrid (Spain)	€ 1,200 + VAT
June 4–6, 2012	Amsterdam (Netherlands)	€ 1,200 + VAT

Specification by Example: from user stories to acceptance tests

by Gojko Adzic

Date	Place	Price
March 22–23, 2012	Berlin	€ 1,200 + VAT
Apr 30–May 1, 2012	Helsinki (Finland)	€ 1,200 + VAT

Website:
http://www.diazhilterscheid.de/en/knowledge_transfer.php

Define test strategies.

Identify efficient test strategies for different areas:

1. How to find out what to test, the use of test automation to reach the desirable test goals?
2. Which test techniques and tools might be useful?
3. How to select test data?
4. How to build and configure the test environment?
5. How to prepare and run the tests?

Specify test metrics.

Which test metrics are to be used for follow up of the testing and the associated goals?

Specify the "Definition of Done" related to test and quality.

1. Prepare "Definition of Done" (DoD) as a checklist by the team to determine when activities/tasks are performed?
2. Checklist of the task that needs to be completed before declaring testing as 'complete'?

Stopping Point.

1. When to continue or stop testing before delivering the system to the customer?
2. Which evaluation criterion 'must' be satisfied?
3. What final acceptance criteria must to be satisfied?
4. The Full Lifecycle Object-Oriented Testing (FLOOT) method – Agile way

The Full Lifecycle Object-Oriented Testing (FLOOT) methodology is a collection of testing techniques to verify and validate object-oriented software. The FLOOT lifecycle is depicted in Fig. 4, indicating a wide variety of techniques. Although the FLOOT method is presented as a collection of serial phases, it does not need to be like this. The techniques of FLOOT can be applied with evolutionary/agile processes as well. It forms a valuable agile testing method, too.

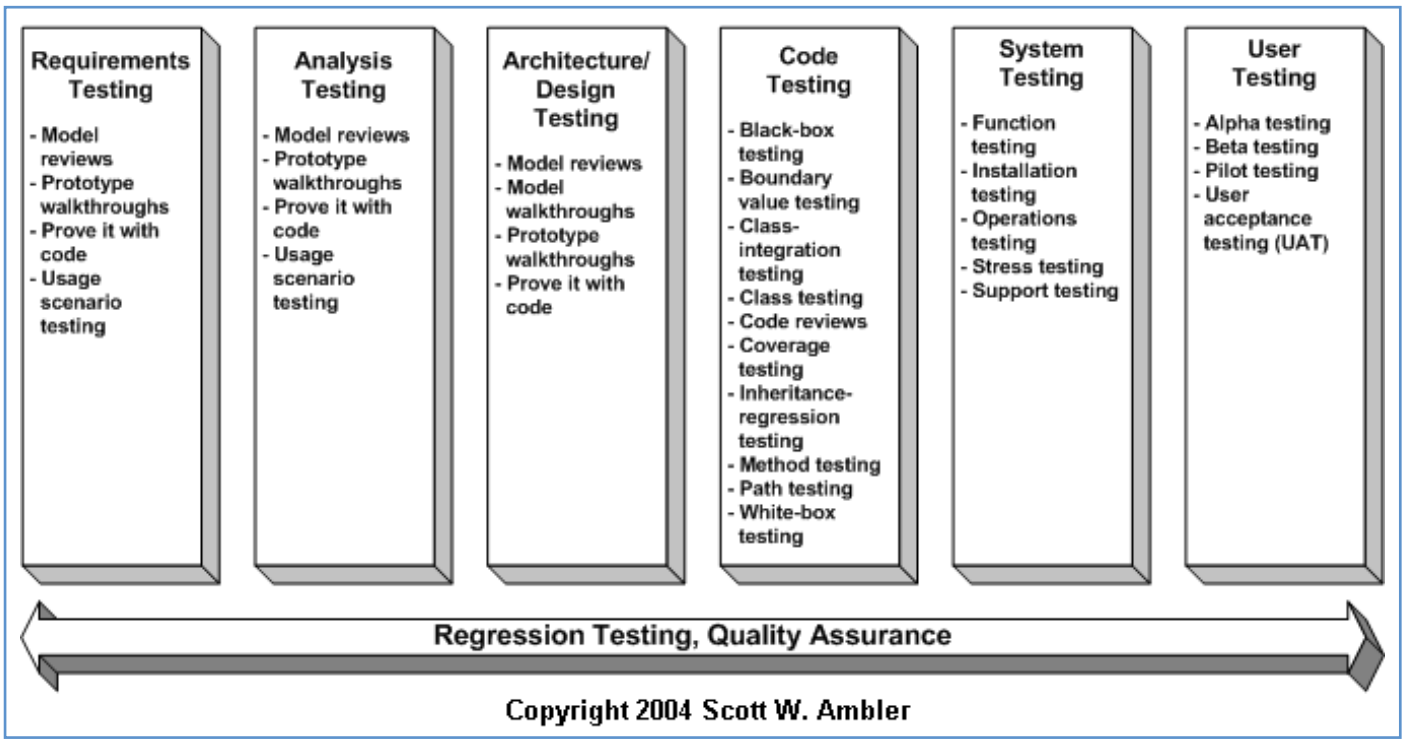


Figure 4 :The FLOOT Lifecycle.

Listed below are FLOOT techniques. The selection of the technique depends on the requirement.

1. Black-box testing
2. Boundary value testing
3. Class testing
4. Class integration testing
5. Code review
6. Component testing
7. Coverage testing
8. Design review
9. Inheritance-regression testing
10. Integration testing
11. Method testing
12. Model review
13. Path testing
14. Prototype review
15. Prove it with code
16. Regression testing
17. Stress testing
18. Technical review
19. Usage scenario testing
20. User interface testing
21. White-box testing

Example:-

Requirement:

As a customer

I **should be** able to login via the web page

so as to place and order for a book.

For this requirement

1. Model design,
2. Boundary conditions,
3. Prototype review,
4. User interface testing,
5. Regression test,

are a few techniques from FLOOT. The agile test cases can be prepared accordingly.

Depending on the requirement, the FLOOT technique can be customized to prepare acceptance criteria for each of our requirements based on the customer requirement.

5. Conclusion

Although Agile being the preferred methodology for deliverables now, there is still scope for improvement. With testing being a vital phase, various FLOOT techniques can be useful for achieving quality of the requirement and keeping to the timeframe. This article was mainly concerned with how to implement agile methods and what questions and rules need to be answered/followed prior to applying agile methods blindly in any project.

6. Reference

[1] http://en.wikipedia.org/wiki/Agile_software_development

[2] http://www.ambysoft.com/essays/agileTesting.html

[3] http://www.eurostarconferences.com/media/70933/315_ebook_anders_finalnew%20with%20reference.pdf

[4] http://www.ambysoft.com/essays/floot.html

> biography

Hemangi Gawand
has nine years of experience in various domains, such as embedded wireless, building automation, image processing and in the telecom. She has industry experience with TechMahindra Ltd., Mumbai, and Patni Computer Systems Ltd. (currently iGate Patni), Mumbai. She was previously associated with the Indian Institute of Technology, Bombay for her research work. She holds a B.E. - Electronics and an M.E. - Electronics as well as a communication degree from Mumbai University with specialization in wireless telecommunication. Currently, she is pursuing her PhD. She is connect via hemagawand@yahoo.com

IT audit and software testing: Future Partners in quality

by Cordny Nederkoorn

Recent security breaches (Sony, DigiNotar, Dutch municipalities) have shown that online applications from enterprises and government agencies are not as secure as they are expected to be. They have their weak spots and crackers (hackers with criminal intentions) know how to exploit these for their own benefit.

How can a web-based company prevent such a security breach? To answer this question we have to see where these weak spots and possible resulting security breaches occur.

Weak spots can be found on process and on operational level. For both levels different professionals are needed to look for the weak spots. Let's start on the process level.

Due to (inter)national compliance legislation companies have to undergo a periodic comprehensive review of an organization's adherence to regulatory guidelines.

This is the field of IT-auditing. Regarding the security *process*, auditors review security polices, user access controls and risk management procedures over the course of a compliance audit. The examination in a compliance audit will vary depending upon whether an organization is a public or private company, what kind of data it handles and if it transmits or stores sensitive financial data.

However, an IT-audit does not necessarily take in account the quality of the security *operations*. For this field security testers are necessary.

Why are both types of professionals needed? I will demonstrate this with an example of a security breach which happened recently in the Netherlands.

DigiNotar was a Dutch Certificate Authority (CA) owned by VASCO. On September 3, 2011, after it had become clear that a security breach had resulted in the fraudulent issuing of certificates for different governments, companies and organizations, the Dutch government took over operational management of DigiNotar's systems. The company belatedly notified its customers that the breach had taken place, so lost its credibility, and with it its main reason of existence.

The same month the company went bankrupt. How could this happen and how can we prevent it in the future? In order to issue certificates to other parties, CAs have to conform to policy guidelines, in accordance with legislation. Different organizations like ETSI and Webtrust develop these requirements together with the CAs.

DigiNotar was apparently accredited conform to ETSI 101 456 (policy requirements CAs) and WebTrust CA - Extended Validation (EV) Certificates standard. Therefore it was allowed to issue EV SSL certificates, which are automatically recognized by browsers, identifying it as a trustworthy website.

The examination for accreditation was done by an IT-audit company. As already said, IT-audits examine the process, by reviewing the company's security policies. In order to examine if a hack had taken place at DigiNotar, the Dutch government hired Fox-IT, a Dutch IT forensics company, to investigate further. Its interim report, ‚Black tulip', showed DigiNotar's systems were already breached in June 2011 and five main security findings on operational level were reported. These security findings are mentioned in table 1.

Table 1: Preliminary findings Interim report Fox-IT ‚Black Tulip'

Findings 'Black Tulip'
No anti-vrus software present: malicious software found
Separation critical components not in place; access via management LAN
outdated and not-patched software on public web servers
No secure network logging
no adequate authentication process for CA-domains

Studying the ETSI guidelines (baseline police requirements for CA operational and management practices) I noticed all five security findings should have been found during a periodic audit. Therefore accreditation was inaccurate and DigiNotar should have been disqualified for issuing digital certificates.

So, something went wrong during this IT-audit. How can this be prevented in future?

In my professional opinion, improvements can be made to enhance the IT-audit.

First, the frequency of IT-audits should be higher. Now it's a yearly checkup, why not make it a quarter-yearly checkup?

Second, the 5 security findings were on operational level. As already said, IT-auditors look at the process level. To ensure the robustness and secureness of the company's IT operational systems, an IT-auditor should work together with an independent security tester (at least CEH, Certified Ethical Hacker). A CEH knows how to detect and report security breaches on operational level. This person can develop and execute specially developed regression tests (acting at the weak spots) to locate possible security breaches.

Third, to avoid miscommunications, regular meetings should be held between the CEH, the IT-auditor, the company's security officer and the company's management. This not only avoids miscommunication, but also keeps security on a high priority in the company's management.

And last, a CEH must have 24/7 access to a central security network logging server that monitors if security breaches take place.

If these occur, a notification has to be sent to the company's security officer, the company's management and other stakeholders like (affected) clients or government. This notification should at least report the time the breach occurred, the place of breach, the possible cause and the measures taken to deal with it. Mind you, to establish such a process is not an easy and inexpensive job.

It will take time, effort and money, but the Return of Investment will be high.

Security breaches will be found, fixed faster and, what's even more important, the transparent process increases the trust of the clients in the company's IT infrastructure. No more security by obscurity, which can lead to non-transparency, resulting in negative media exposure and worse.

This example showed a scenario where the future co-operation of two professions, IT-audit and security testing can improve the security of the company's infrastructure and IT-processes. IT-audit and security testing operate on different levels, but have a similar goal: to examine quality!

References

Certificate Authority: http://en.wikipedia.org/wiki/Certificate_authority

IT-audit: http://en.wikipedia.org/wiki/Information_technology_audit

DigiNotar: http://en.wikipedia.org/wiki/DigiNotar

ETSI: http://www.etsi.org/WebSite/Standards/Certification.aspx

Webtrust: http://www.webtrust.org

Fox- IT report ‚Black Tulip': http://www.rijksoverheid.nl/bestanden/documenten-en-publicaties/rapporten/2011/09/05/diginotar-public-report-version-1/rapport-fox-it-operation-black-tulip-v1-0.pdf

Certified Ethical Hacker (CEH): http://www.eccouncil.org/CEH.htm

> biography

Cordny Nederkoorn
has worked as a software test specialist for the past six years. He has extensive experience in (automated) system testing of online applications for the financial and government sectors.
Currently, Cordny works as a software test specialist at Immune-IT, a Dutch test consultancy firm, where he develops and executes system and security tests for his clients.
In his spare time he is active in different testing and identity communities, reviewing and testing internet protocols for online identity and access management, an area in which he wants to further practise his security testing.

iTesting – "less bored, more inspire. Less donkey work, more games. Hate less, fall in love"

by Monika Pluciennik

On that day I managed three testing projects, having just a short break for a morning tea, then back to the testing world. I was so focused on the work and switched off from the external world that on the way to the car park I understood nothing of what my friend told me that the CEO of Apple had passed away. My thoughts were still on the projects, so I just asked "What are you talking about?", not sure why he was sharing news about somebody's death. He looked at me in astonishment that I had not heard about anything. Steve Jobs died and the world lost a visionary and creative person who had a big impact on our world, not only on the IT market, but the world society and the way we communicate with each other.

In these days communication is about iPhone, iBook, iPad... A couple of months ago I thought we didn't have fun during performing testing; it was a difficult and time consuming job. Why not to make it one of the "iProducts" that are so successful. Make testing an **iTesting**.

Please let me to follow the words of Steve Jobs from Apple: ***"We have always been shameless about stealing great ideas"*** and present what I mean by iTesting ***"be bored less; inspire more. Less donkey work, more games. Hate less; fall in love"***.

I - stands for "I love testing". Why?

In the future, proper testing will not be possible without people. Most of the testing will be automated, but the tester will guarantee that correct testing will have been performed. You are not able to be a good tester, if you don't fall in love and keep the relationship going strong every day. So as Jobs said: ***"When I hire somebody really senior, competence is the ante. They have to be really smart. But the real issue for me is "Are they going to fall in love with Apple?" Because, if they fall in love with Apple, everything else will take care of itself. They'll want to do what's best for Apple, not what's best for them, what's best for Steve, or anybody else."*** Just replace the company name (i.e. Apple) with Testing and you have the idea why you should fall in love.

T stands for '**The Tools**', of course future testing tools. How they are going to work? ***"Design is not just what it looks like and feels like. Design is how it works"***. Would you like to have tools that would do at least 50% of your manual test activities, like reporting, creating requirements and test cases, test sets for executions? In my opinion, 30-40% of test case preparation can be done by using an automated test library.

Example: you need to log into web applications. Depending on the company policy, you always need login credentials and just numbers of credentials; types of them and validations around them are changed frequently. The same situation we meet around exporting functions from web applications: just source data, export formats are changed. Imagine a large test case library that you may link to the requirements and your basic test cases are generated.

E stands for '**Effective Testing**'. The new future tools will improve the effectiveness of testing significantly. What do I mean by that? Twenty years ago I would not immigrate to Australia, as I wouldn't be able to communicate effectively with my family in Europe. Now, the communication is about ***"Click. Boom. Amazing!"*** Even my seven year old son uses an iPod to tell his grandmother about his daily activities. So why not think about the tools that allow us to be so effective that, the basic testing is done immediately, as soon as a new code is released; like Click = automated testing, Boom = the list of defects is created automatically, Amazing = we have already done by now about 50 % of all our planned testing tasks. This final result cannot be achieved without tools in which the artificial intelligence of testing will be applied. There should be also a direct connection between the application under test, the code and the test cases. So for example if a defect is found, the developer will be informed exactly which line of code caused the problem, even if the application was using middleware and data were sourced from multiple backends. Then during the fixing, defect information will be given back to the tester, specifying which test cases need to be rerun to ensure the correct regression testing will be chosen. The same will be applied to requirements: if the existing requirement is going to be changed, the info will be linked to the code lines which need to be changed. That will provide links to existing test cases that need to be modified and rerun. Test management tools will have the reference between the requirements, the test cases and the relation between them.

In the current stage of knowledge, we miss a link to the code and I'm sure that in the future when systems can store much more information this can be fully automated.

S stands for '**Smart Testing**'.

Have you heard about S.M.A.R.T. (Self-Monitoring, Analysis and Reporting Technology; often known as just a **SMART**)? This is a monitoring system for computer hard disk drives to detect and report on various indicators of reliability, in anticipation of failures. Why not create something similar for any testing project? It will contain standard tests that need to be run in particular types of project, standard reports, documents that will gather information from the other applications, standard metrics, automated check lists. Obviously, without future testing tools this will be not achievable.

T stands for '**Talk About Testing**'. The whole community, not only people involved in the SDLC, will perform testing. Some of the companies outsource testing at present. In the near future, I think everybody will have the possibility to perform testing. Of course, this does not mean that each human being will be involved. I expect that companies will advertise testing tasks on the web, and - like at eBay - who offers the best rate will get the chance to do the testing. Testing will be done by multiple people on different platforms and languages. Quick feedback will be available straight away, and this will be real world testing, not just testing done by company testers or outsourced testing.

As we are faced with iProducts everywhere from buying the bread in a market to buying medicine at the chemist (did you know that the chemist is an occupation, which will disappear from the job market due to robot replacement?), everybody will be responsible for picking up any "production issues, improvements". I have not mentioned about the defects, as these will be always automatically reported to the software vendor with 5 "W" keywords about defect information – who, when, where, what and why.

I stands for '**Innovation Based Testing**'. To make continuous improvement, future testing tools will need a lot of innovations. The innovations will be required not only within testing but in the whole SDLC beginning from the phase of requirement gathering, and ending on production software support/decommission. The creativity and innovative ideas will be the most desired skills of the test lead as per Jobs "**Innovation distinguishes between a leader and a follower.**" Continuous improvement will allow focusing on more inspired parts of the testing job and will lead to improving product quality.

N stands for '**Networking Testing**'. I think that in this phase of the future testing we are right now. We have a lot of forums, web seminars, journals which are available for everybody for participation and sharing knowledge. At this stage, most of them are from definitions point of view and, definitely, there are too many of them. In the future, only the strongest of them will be present on the web market. Possibly ten, but not more than twenty, so finding information, groups will be much easier. Also when the basic testing knowledge will be covered by everybody (meaning that you have a foundation level testing certificate), we will move into sharing the real life cases of the testing issues that we have in our environment and the solutions that we implemented. There will be a shift to the case scenario. Example: At the SGIST conference in Melbourne, one presentation was about data management, when another referred to the practical problems of transformation testing in the food industry. So this community trend will be very important in future testing.

G stands for '**Game-Like Testing**' with the prediction that you are going to achieve your goals. Imagine that the project you are working on is like a game. Your requirements will be your treasures which will be accessible when you execute test cases. The test progress bar will be like in the game, you will know how many tests you have done, how many left and what will happen if the change is put into software code. The estimation (current state) is calculated automatically and if you are on time you still have credits available if your conditions do not worsen. Using future testing tools you will be able to make your own customizations that you need from your test management point of view. This should be done as described by Jobs ***"We made the buttons on the screen look so good, you'll want to lick them."***

I suppose that iTesting will be like ***"I think we're having fun. I think our customers really like our products. And we're always trying to do better"***, and this will lead to the improvements in the quality not only of our products but also of our lives. Thank you for letting me share my imaginations with you. Just remember that ***"everyone here has the sense that right now is one of those moments when we are influencing the future"***. For me, the future of the testing is less bored; inspire more; less donkey work, more games; hate less, fall in love with iTesting.

*All quotes by Steve Jobs

> biography

Monika Pluciennik
has over 13 years' experience in IT as a consultant, a business analyst, a tester. Since 2006 has worked in Computershare as tester, senior tester and now as a test lead. Her experience is in Windows testing, web testing and automated testing. She enjoys travelling, especially in Australia.

The future of software testing: What about test management?

by Gert-Jan van den Ham

In our daily life things change. Sometimes slowly. Sometimes we don't even notice. And sometimes we must rapidly adapt to changes because the environment changed. And it requires full flexibility for a change. Even if we don't want to, we must. If we don't change, don't adapt, we fail.

It is our human nature that we change things. That we stay curious how things work or how we can improve things. In history people have always been obsessed about flying. And many people took big risks in the past when they took a chance to fly. The first persons did not have the knowledge we have today. All their failures, however, contributed to the development of our first airplanes. Imagine the first person with wings attached jumping off a hill. Convinced it would work. Or imagine the astonishment of the first people who did manage to fly. To share and publish information is important if we want things to happen. Leonardo da Vinci was obsessed by flying, but he never published his designs. So he has not contributed to the development of airplanes. Sharing your ideas makes the difference.

Let's take another example: the first car. There was no ignition key, there were no airbags, and top speed was irrelevant. When cars became a major way of transportation, we had to change our environment. We had to set up rules: speed limits, major roads and traffic lights to regulate the new way of transportation. And the car industry had to adapt their designs to our demands. The car industry is innovative. 25 years ago everyone was familiar with our way of transportation, but the first demand for huge changes was there. Pollution was the cause for this demand for change. And although we had never heard of an electric car, today the first full electric cars are in production. And hybrid cars are well-known for over ten years.

All things change. What is new today can be old-fashioned by tomorrow. Even things we can't imagine how to survive without will disappear one day. 25 years ago we had our television mainly for entertainment. We had our radio for music, and the daily newspaper for the latest news. We could probably not imagine life without them. Television was the evolution of the old newspapers, radio and the cinemas. Together they gave us information and entertainment. Ten years ago we probably could imagine that personal computers would replace, or at least take a big share in the way we entertain ourselves. And today the personal computer and even the laptop are under pressure. Personally I read my daily news on my smartphone. I listen to music on my smartphone. And my smartphone even replaced the old maps I used to have in my car to find the way. My smartphone has full European card coverage. For Internet browsing I use a tablet. I never go to a cinema, I have television on demand. And I rarely visit my local bank. I have Internet banking, accessible via my laptop, tablet and even via my smartphone. And I predict that in a couple of years we don´t even use the Internet as we do today. The basics will probably stay the same, but the way we gather our information will change rapidly.

Not so long ago, IT was introduced to the normal human working life. A big change in the way we work. Computers replaced typewriters. And automated processes replaced the way we were used to work. IT was the solution for everything. However, it also had its difficulties. Users threw their demands over the wall. Developers threw their solutions back. And in the end no one was happy. So we improved in several ways. Development methods were introduced long ago. Traditional Waterfall methods became popular. Software can makes things complex. When we became aware of all the risks we introduced with all the new inventions, the demand for structural testing and quality assurance was born. Someone independent had to take a look at what was produced by the development team. The big difference between having a look at what was produced and testing is of course the structured way of testing. Testing to find faults, or testing to prove quality, the structured way makes the big difference.

In the traditional projects the test manager was happy if there was some time left at the end of a project. Some time to have a look at the quality. And the testers started a bug hunt. Throw it all back to the developer! And this game continued for some time. If we were lucky, it stopped when the desired quality was achieved, but mostly when time was up. The tester was somewhere between the customer and the delivery team.

In this traditional way of project approach the test team is completely separated from the delivery team. To ensure independence! Testing is a project on its own within the project. We have several roles. Test advisor, test manager, test coordinator, test

analyst, test executer, etc. In a traditional way of development you need all these roles. The test manager is responsible for the entire testing project and plays the game of testing: Starting by creating awareness, creating time for testing. and so on. In the delivery team, however, things are completely different. There is a project manager, but no information analysis manager, nor a development coordinator. The project manager is responsible. There might be seniors to make sure the work is done in time, but I have never seen all testing roles described in all other specialties involved in a project... So why is this? Is it because we are still finding out the best way to deliver something good? Are we standing with wings attached on the top of a hill?

Today we see more and more projects using a different approach. We involve customers. Not as we were used to, but really involve the customer. To make sure the customer gets what he wants. We shift away from the traditional way of development. And we are making the move to the agile approach. All disciplines are involved. And we test together with the end users to prove we made what they want. In these teams we focus on quality - as broad as it gets. Is this what we need? And is this what we want? All other industries work somewhat like this! If we build a car, sales and engineering are closely working together, to make sure the customer will buy the car. If we build a house, the architect, construction worker and the customer are constantly communicating. To make sure the house is what the customer wants.

So if we create the end product together with the customers, developers, information analysts and testers, why do we need a test manager? The same goes for other traditional roles by the way. In the new way of developing software solutions, however, testers will specialize. Depending on the demand of the customers, there will be a need for a specialized tester, just like developers have a certain experience in developing software.

In the future these teams will deliver a product that can be maintained. Software is documented, requirements are written down as story lines and test scripts are delivered. There will be no projects with two weeks left for testing, with test managers left to create awareness for the risks taken or to make a master test plan, or with test coordinators who make sure all tests are performed in time. When the work is done, it's done.

So, now what? What is left for the test manager? Teams will be self-steering. Testing becomes a non-questionable part of development. There will be no discussions about time for testing. Testing is part of development. Specialize, for example in risk management, or whatever is close to our current work as test managers. For the time being, we are still changing for the better. Do not stand still though, because who stands still will be forever on top of the hill with wings attached.

> biography

Gert-Jan van den Ham
started his career as a troubleshooter for a big PC consumer company. With 9 years of experience in this industry he made a career move to testing. He built his own test team at the .NL domain registry. As a test manager he led several test projects with great success. Quality to market and time to market improved drastically. To increase the amount of releases and the quality of releases, he led several Agile based projects. To share his experience, he recently continued his career as test consultant. Currently he is an advisor at several companies to help to improve the test process. His interests are in creating awareness of risks, improving the test approach by looking out of the box, and testing in an agile environment.

The cutting edge

by Bert Wijgers

Traditionally, testers operate on the cutting edge between business and technology. Since the axe of software development used to be very blunt there was plenty of room for everybody. As the axe gets sharper, most testers will have to choose sides: business or technology. The cutting edge will accommodate only a few; the rest of us will be sharpening the axe.

There are lots of ways to deliver bad software and by now we have tried quite a few of them. There are also ways to deliver good software. Let's assume that, slowly but surely, we are getting better at making good software. In the future the right software will be delivered timely and within budget. This implicates that Waterfall projects will become an extinct species.

In agile development, testing and coding are intimately interwoven. Through daily and intensive interaction with coders and designers, testers will learn more about the inner workings of the software. This will change their mindset; testers will abandon the idea of the Black Box. Testing will be guided by knowledge of the code and will inevitably move away from the user perspective.

The agile approach to software development has brought business representatives into the development teams. Testers still play a role in facilitating communication between the business people and the technology people, but no longer in the role of translator. Business representatives interact directly with the designers and coders. Testers can only assist to help clarify misunderstandings.

Another consequence of agile practices is that everybody learns to care about quality. Testers will guide designers and coders on how to check their own work. This will give them time to do other things, like (exploratory) testing.

Specs and checks

An important part of what is now called software testing is confirmatory in nature; test cases are executed to see whether or not the software reacts conform to specifications. Another part of software testing is exploratory; test cases are executed to see whether or not something goes wrong. This is the distinction between checking and testing (Bolton, 2009) and it limits the scope of what testing will be in the future. All testing will be exploratory by definition. Confirmatory test activities will be called checking and they will be an integrated part of the software development process.

Checking can be planned, scripted and automated to the extent that specifications are stable. Unfortunately, specifications are never complete nor ever completely valid, so they change. If they don't, the product will not be able to satisfy the stakeholders. There is enough evidence by now that the idea of "big specifications up front" is just not workable for most types of software. So, work on specifications will go on throughout the project.

The testers of today can be the requirements engineers of tomorrow since they know about the business and they know about the technology. On top of that, they are analytical and can express themselves objectively in human language in a clear, concise and complete way. And they know when a spec can be checked.

A check can even be a specification itself, and as such drive design and coding activities. This is now commonly known as Test Driven Development, but in the future we might call it Specification by Example (Adzic, 2011). Software can be checked fast and often by making specifications executable.

The specification and the check will become one thing and because it is executable it is easy to keep up to date. At the same time the check is a piece of documentation. In the future there will be no separate requirement documents, functional designs or technical designs. There will be one description of the software in the form of something that we used to call test cases. These will be automated from the day they are born as examples of specifications.

Looking for trouble

And then, of course, there are the non-functional specifications. Unlike functional specifications those cannot be stated in a straightforward, simple and checkable way. Therefore non-functional testing has a bright future. It is easy to check whether or not a particular screen is loaded within two seconds under particular circumstances, but it is not easy to check whether it does so under all circumstances. The same reasoning holds for security testing and different sorts of "-ility testing". Non-functional test-

ing is about looking for trouble and not about confirming that all is well.

Only when it has been checked that the software does what it is supposed to do, in terms of features and functionality, it is ready to be tested. The software that is released for test will get better, but it will never be perfect. The defects will be fewer and harder to find, so the functional tester of the future has to be very good at bug hunting. There is no sure way of telling where bugs are hiding, so traditional testing methods are of limited use. The context-driven school of testing (Kaner and others, 2002) will become more prominent, because it is focused on individual skills.

Traditional functional testers who stick to their scripts and plans will play no significant role in the future; they will fall of the ever sharper cutting edge. If they want to cling on, there is plenty of room on both the business side and the technology side. Projects will benefit greatly from requirement engineers with a background in testing. Good specifications are the basis for confirmatory checking activities. Those checks can be scripted and automated.

Testing can be aided by tools, but it cannot be automated. A human user has to interact with the software to experience its quality. The actual end users can and should do this. Professional testers can take acceptance testing to a higher level, but they can never replace the actual users.

The code and its automated checks are developed by interdisciplinary teams. Before releasing the software, it has to be tested by an outsider. In the future, software testing will be what it has always been: a highly explorative quest for bugs as a last line of defense. The big difference will be in the quality of the software to be tested.

References

Adzic, G. (2011), Specification by Example, How succesful teams deliver the right software, Manning, New York.

Bolton, M. (2009), Testing vs. checking, http://www.developsense.com/blog/2009/08/testing-vs-checking/

Kaner, C., Bach, J. and Pettichord, B. (2002), Lessons Learned in Software Testing, A Context-Driven Approach, Wiley, New York.

> biography

Bert Wijgers
is a test consultant with Squerist, a service company in the Netherlands. Squerist focuses on software quality assurance and process optimization. Its mission is to inspire confidence through innovation.
Bert holds a university degree in experimental psychology for which he did research in the areas of human-computer interaction and ergonomics of the workspace. He has worked as a teacher and trainer in different settings before he started an international company in web hosting and design for which a web generator was developed. There he got his first taste of testing and came to understand the importance of software quality assurance. Bert has a special interest for the social aspects of software development. He uses psychological and business perspectives to complement his testing expertise.
In recent years Bert has worked for Squerist in financial and public organizations as a software tester, coordinator and consultant. He has written several articles about testing and is a passionate speaker on topics related to software quality.

Futurist View of Testing

by Wayne Ellis

The thing that is great about testing is the adaptability of the techniques learnt, which can be applied to almost any project that requires testing. New techniques or methodologies may come along which may tweak or rearrange the way professional testers carry out their daily tasks, but nothing they should not be unable to handle. There is the science of predicting the future called futurology which many people from different backgrounds (scientists, authors, and social scientist) and companies are involved with. In predicting the future you can predict where to head your company towards, or what pitfalls/issues/risks the future may hold. One such company is the chip maker Intel who have a resident futurist, and have recently published four short stories from science fiction writers which have used technologies that Intel are working on in their labs at the moment.

So in predicting the future of testing I have turned to the predictions of technologies outlined by the science fiction writers. Some may say that the science fiction authors have been predicting the future for a while now and some have got some predictions right, Arthur C Clarke, William Gibson and George Orwell to name but a few.

Cyberspace

This term is used today, but in the future it takes the internet to the next stage. It was predicted the cyberspace of the future would be a virtual environment that is controlled and viewed in the user's mind. This would be through a deck (e.g., a computer) connected to the network and connected to the user through a neural implant.

So, a browser in the brain, you will be testing the software integration behind the deck and the network, you also have to make sure that the software has no side effects on the users themselves. It would become almost mandatory to carry out clinical trials before the software goes live. How many times has your web browser crashed on you, what happens when this occurs inside your mind?

As already seen from the news channels with more and more people, companies and data online, the influx of malicious viruses and hackers, security testing is gaining more and more testing attention. In this future everything is accessed from the cyberspace, and companies will want to make sure their security protocols are up to the task. Described as ICE programs (= Intrusion Countermeasures Electronics) in the books, these are your firewalls and virus sweepers. Some corporations decide to take the internet law in their own hands and employ BLACK ICE. This is a program that will not only try to stop the unauthorized access to the corridors of the corporation's cyberspace area, but track and launch its own attack against that hacker, be it finding their location, or wiping their computer, or in some cases causing harm to the connected user. Again, this is another layer of testing:, the software's interaction with the real world.

Artificial intelligence

While today AI exists in a simple form, we will in future be almost reliant on AI to carry out everyday tasks that either too complex for us to manage or too simple for us to bother about. AI will consist of almost self-aware programs that will be able to respond to questions like a human. So when it comes to testing, what is there for us to do? Would another AI be programmed that would be the tester, and this AI's job is to test other AIs, or would we have to interview the AI like a person to ensure it's up to the job? What is most likely is that the AI will be placed in a test environment that is a copy of the live environment and left to run with us monitoring its progress and introducing test scenarios to see how it handles them.

The one theme that is always highlighted about AI in books, irrespective of whether the AI comes in the form of a machine controlling a space ship or is placed into a robotic/android body, sooner or later it will try to take over the human race. Isaac Asimov offered the creation of three laws of robotics, which is to say that, to stop the robotic apocalypse, these must be among the top tests carried out by the future testers to ensure the continuation of the human race.

1. A robot may not injure a human being or, through inaction, allow a human being to come to harm.
2. A robot must obey the orders given to it by human beings, except where such orders would conflict with the First Law.
3. A robot must protect its own existence as long as such protection does not conflict with the First or Second Laws.

Biological machines and nanotechnology

Nanotechnology can simply be described as tiny robots programmed to carry out a task which they will do, be it repair skin cells after a cut, destroy cancerous cells, or remove atoms from materials to destabilize and break them down into their smallest components. Prince Charles of UK asked the Royal Society to investigate the risk of these machines running amok and turning everything into "gray goo". Once again, its software has an effect on the physical world through a tool, so testing will have high priority to ensure that there are no bugs in the system that could cause the machine to destroy what it should save.

Biological machines are the pinnacle of organic computing. Here the users would use cells as machines that they could program to produce different results. Using DNA as the machine language the program would be written out and entered into the cell and the results would be returned, for example a program of producing more hair follicles, or burn more energy to reduce fat reserves. Even though this is not a computer the testing principles stay the same: you have a program, your input and expected output, just the type of environment would be different.

References

1. http://techresearch.intel.com/tomorrowproject.aspx Intel's Futurism information page
2. http://www.engadget.com/2011/10/25/the-engadget-show-026-a-visit-from-intel-a-trip-to-new-york/ Interview with Brian David Johnson Intel's resident Futurist (starts after 28.00 mins)
3. http://www.bbc.co.uk/news/technology-15483792 BBC news on predicting the future.
4. http://www.nanotec.org.uk/finalReport.htm Royal Society investigation into effect of Nano machines.

> biography

Wayne Ellis
having passed his ISEB foundation and practitioner for software testing has being working for Capgemini as a senior test analyst. He has 7 years specializing in data warehouse testing. He was once described as somebody who "... doesn't think just outside the box, he is thinking in a different room to that box altogether".

Agile Development Lifecycle Will Change The Way Of Traditional Testing Methodology

by Melvin Chua

As more organizations embrace the faster paced development lifecycle of methodologies such as Agile, the role of the tester is bound to change. Traditional requirement-based testing techniques, such as state based transition testing and use case testing design techniques, will be less frequently used and replaced with more agile testing techniques, such as risk based testing and exploratory testing. With user stories replacing comprehensive requirement documents, testers will need to adapt and change the way test cases are designed as the test analysis and design phase will have a shorter time span.

Test automation is another area that will play a crucial part in the agile framework. In a fast paced environment like Agile and with a need to do regression testing of previous sprints, automated test frameworks will be great since they allow that we can reliably replay the automated scripts without it failing.

New generation test automation tool

Test automation in general has never had a good return on investments for the organizations that have embraced it. I believe that this will change with emerging new test automation tools where programming skills are not paramount and where front end Graphical User Interface (GUI) screens and wizards will drive the test automation framework.

The reason why test automation rarely succeeds is, in my opinion, the cost associated with maintaining the code that drives the test automation. Whenever an application goes through maintenance and the code of the application is updated, so too does the code that drives the test automation. The code would then need to be tested and debugged if it doesn't work properly, which effectively makes this a development lifecycle rather than a testing lifecycle. This is very time consuming and rarely do testing projects have the time to maintain a test automation framework.

If the test automation framework is designed from a graphical user interface and testers only have to fill in the test data and actions for automation without having to debug and test the code of the test automation framework, this will reduce the test automation time significantly. New generation tools such as Tosca are already achieving this to some extent.

One of the benefits of test automation is the significantly reduced time of test execution as an automated test can replay the script in less than half the time it takes to run a manual test. This suits the agile lifecycle, as the testing of previous sprints which have previously been tested manually can be very time consuming.

Another change I see happening is that business domain testers will need to improve their skills and be more technically minded. Whatever tool is used for test automation, the same principles of good test automation design is important. This requires some form of technical competencies, as a poorly designed test automation framework can lead to painful and long periods of rectifying the test automation.

Agile Manifesto and testing

As testers the two principles of the Agile Manifesto that I think will change the way of traditional testing methods are:

- Working software over comprehensive documentation
- Responding to change over following a plan

In order to overcome the lack of detailed requirement and to produce working software, testers will have to rely on business analysts and subject matter experts to determine the test oracle for the application under test. User stories often contain inadequate information for the test oracles to be determined. This is where the use of exploratory test charters to document the test execution over following a pre-scripted test script would be beneficial. The results of the exploratory test charters can be discussed and verified by the business analyst. The beauty of exploratory testing is that we can begin testing with a minimum amount of test documentation and guide the testing and document the results as we go.

The other traditional test method I use is to write a detailed test plan at the beginning of the project during the test planning stage. With an agile lifecycle where change in every sprint is possible, writing a detailed test plan right at the beginning of the project may render the test plan completely obsolete as the project goes on. Testing needs to respond to change over following a plan. To this end the test plan needs to be brief and be easily changed when the need to do so arises. Alternatively, testers

could write a brief test plan for every sprint with just enough information to guide the testing through each sprint. Test objectives and test milestones for each sprint would need to be clearly distinguished if testing is to add any value to the agile project.

With the focus of Agile on "just enough documentation" rather than detailed documentation, standards such as IEEE 829 would become too heavy handed. The challenge for the testing profession is to get the balance right between how much information is just enough and how much is too much information. After all, testing is about communicating the quality of the application under test to the project stakeholders so that they can make an informed decision on whether the application is ready to be released into production. Too little information and the risk would be that the project stakeholders make a wrong quality decision based on inadequate information.

With more organizations demanding shorter time frames for delivery of Information Technology projects, testing is often the component that gets shortened the most, as it is the last step before shipment to production. It is on the critical path, and the tester is under pressure to deliver in a very short time span.

The agile development lifecycle has proven that it can deliver applications into production in a relatively short space of time. Testing needs to adapt and change with the project. The tester would need to identify the right techniques to get the job done. Change can be unsettling, but if the tester is equipped with the right tools and techniques, the fast paced challenges can be managed. As more is demanded from the testing team in less time, the Test Manager would need to manage resources and come up with a plan to work smarter with the help of the right testing tools. The test teams that are going to succeed in such an environment are those that will work together, where the entire project team supports the testing effort with the test professional guiding the project team in the right direction, especially if non-testers should be called upon to help with the testing effort.

> biography

Melvin Chua

I gained my Diploma in Business Computing in 1999. I have been working for the Wellington City Council as Senior Test Analyst for the last 3 years. I have worked as a Test Analyst both for the private and public sector in New Zealand for ten and a half years, and I am certified as ISTQB Advanced Level Test Analyst.

Testing Mobile Devices – A Journey to the Future

by Attila Fekete

Let's start with a controversial question. Do today's most successful companies satisfy or create customer needs? The word 'create' might not be the right one as this is about giving reasons for customers to buy something they would not think they should otherwise buy. Most probably the question which one dominates is just as controversial as the original question itself. However, as far as today's technology leaders are concerned, the generated need dominates. There is no surprise here as most people are simply not aware of the technology advancements and ongoing research programs. Most of them got used to the limitations so much that they forgot to dream. So how could one expect them to come up with new futuristic needs? And if this is how it goes, then the technology leaders and visionaries are the ones who create new trends. Along with the new trends they challenge us quality professionals with ways we have never seen before. So to some extent we are in total control of the challenges we are going to face in the future. It might sound weird, but think about it! The reasons why we are keen on complexity and more and more advanced technologies is most probably psychological and motivated by our desire of immortality, but this is not the subject of this article. To have a complete picture of the future of software testing, removed from any product trends, one should also consider advancements in the quality engineering field.

In this article I am not trying to predict how the world will look like fifty years from now. What I will try to focus on are mobile devices and the way they will change how we and everything around us will communicate. This is most probably one of the hottest areas in the coming years as Ericsson's CEO predicted 50 billion connected and communicating devices by 2020. As we are talking about the future, please do not take everything that is mentioned in this article for granted, but consider these elaborations as the result of a brainstorming session. Maybe certain ideas will never reach mass production, or maybe even some of my ideas will be far exceeded by reality.

So let's start our journey about the possible features of future mobile devices paired with the challenges they will put on us.

Display:

You do not have to be a visionary to be able to foresee that 3D will be a staple feature in future mobile device releases. So what even more advanced feature is still to come, one would ask. Having seen Samsung's pico projector phone at the Mobile World Congress in Barcelona last year, you would be amazed how they could equip such a thin device with a projector. Although the technology still needs some polishing, I'm sure you won't have to wait too long. And if you accept that projection will be a standard feature, then it is not a flight of fancy to assume that holographic projection will arrive one day, too. Here I can hardly imagine anything other than manual testing.

Input:

As the mobile devices will be equipped with 3D holographic displays, they will have 3D desktops with XBOX's Kinetic-like control where the desktop content will be displayed in 3D and the user's hand-movements will be detected and converted to control signals. For example, the user will be able to grab an item displayed in 3D and move it aside.

The other way would be, instead of using hand movements, to be able to control the mobile device by brain waves. You would be able to call someone from your contact list by simply thinking of the person's name you would like to call.

Future devices will also offer augmented reality replacements of traditional human input methods. Even touch screens and touchpads might disappear and will be replaced by a projected holographic one.

Without any doubt, voice control could also be an option for interacting with a mobile device. We already have this feature in some of today's high-end devices, but it is far from perfect and not a standard feature either.

No matter how users will interact with their favorite toy, unavoidably the long-buried manual testing will put up another fight against automation for a while. At least for testing the Kinetic-like and voice control features, manual testing will be the only option. I presume manual testing will be the only option for testing the brain control too, but maybe I am completely wrong and one will develop an automation framework which will be able to mimic human brain waves and make manual testing obsolete in this area, too.

Prof van Testing recommends

Díaz Hilterscheid

IREB -
Certified Professional for Requirements Engineering - Foundation Level

Description

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

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

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

IREB International Requirements Engineering Board

Dates*	3 days
23.–25.01.12	Mödling/Austria (de)
28.–30.03.12	Berlin/Germany (de)
06.–08.06.12	Berlin/Germany (de)
14.–16.02.12	Stockholm/Sweden (en)
16.–18.04.12	Helsinki/Finland (en)
19.–21.06.12	Oslo/Norwegen (en)

*subject to modifications

Website:
http://training.diazhilterscheid.com

Middleware:

Future mobile devices will be rather more adaptive than today's devices. They will be shipped with a context-aware middleware and have a cognitive computing chip inside their diversely-shaped bodies thanks to nanotechnology. They will not only adapt to the context like time, location, weather conditions, day of the week, etc., but mobile devices will learn about your habits and use this information to make your life easier. There is definitely an information boom that has been going on for a few decades. Future mobile devices must filter and efficiently present this huge amount of information, the right information at the right time. For example, if you activate out of office in your calendar, that you are on vacation and going skiing, then you are probably interested in snow and weather conditions. When you are driving your car on a highway and heading to a meeting, you are probably interested in traffic jams.

Future mobile devices will also have dynamic user interfaces which will use the information gathered about your habits to customize the arrangement of the widgets on your mobile phone's screen.

If Machine Learning (an area of artificial intelligence research) will be used in a large volume, then testing of software with built-in artificial intelligence modules will become general within testing. This is something that is not the case today. The quality of a Machine Learning algorithm does not guarantee that an application implements and uses it correctly. Test engineers will need to verify that an application using a Machine Learning algorithm correctly implements the specification and fulfills the users' expectations. It is really challenging to find defects in an ML application, as there is no reliable test oracle as such to indicate the correct output for arbitrary input.

OS diversity:

The question is how diverse the mobile OS market will be. As it seems right now, Android will dominate the mobile OS market followed at a distance by Apple's IOs and Windows Phone. For sure time by time new contenders will show up and put up a fight, but looking at actual trends Android's top spot will not be in danger. The divide between desktop and mobile OSs will narrow offering you a seamless working environment. All these look promising from third-party application developers' point of view. OS fragmentation, however, is an issue for application developers. Quite a few mobile vendors have struggled updating the OS with the latest version on devices not sold. Mobile phone vendors need to improve in this area as their sales figures are connected to the eco system built around their products. Third-party mobile application developers can expect a much more homogenous platform market than it was ten years back. For them it will mean less testing on different platforms and this way lower application development costs. However, the OS version fragmentation is a big question mark, but I presume there will finally be significant improvements in this area, too.

Feature diversity:

Besides OS fragmentation, the feature fragmentation of mobile devices will challenge third-party application developers too: different screen resolution, 2D or 3D, memory size, wireless support, etc... I presume that there will be fragmentation forever, as mobile vendors will have different offerings for high-tech freaks and budget users.

Applications:

You must be silly not to see that mobile eco systems will be critical to drive loyalty. Test professionals working in the mobile communication domain, will no longer test single mobile phone software, but the complete package which will contain a huge amount of software modules. And not only the amount of software will grow, but as mobile devices become more and more powerful their complexity will increase significantly, too. This will challenge all those who are working in this domain.

Mobile device owners in the past did not have to worry about viruses as they were simply rare exceptions. With the appearance of the Android OS, this has changed. If we consider that according to predictions by 2020 there will be 50 billion connected devices, which also includes your fridge, then viruses will pose a much more serious threat: they can spread faster and infect even your fridge. Considering that in the future really sensitive information (passport, insurance documents, etc...) could be held within your device, security testing in the mobile domain will soon become critical.

Cloud-based applications will be common on mobile devices, too. One of the pioneers is Apple's ICloud service, which is a really handy addition and shows the biggest strengths of cloud-based services: all your applications and all your data (music, documents and calendar) are available no matter which shiny toy of yours you are using. One day maybe cloud-based applications will come alive and instead of passively storing your data they will evolve through learning your habits. For example, if your favorite singer releases a new remix you will get the demo immediately on your cloud storage and you can access it from anywhere. Test engineers will do more and more cloud-based application testing. As users will store sensitive information in the cloud, security testing will be a key contributor to the success of such services.

Usability:

Ease of use has never been more important than it is going to be in future. At least in the consumer product domain it will definitely be a key factor to catch the consumer's attention. Companies willing to stay in the competition will put more and more focus on usability testing which is manual testing by its very nature.

Battery charging: It is for sure that today's battery chargers will disappear. Either mobile phones will be charged by changing electro-magnetic fields or by the surrounding voice and noise.

So it is certain that manual testing will not disappear and security testing will be ciritcal from a successful business's point of view. We will need to put more focus on usability testing, and we will face new challenges in the form of testing applications with built-in AI modules. Paired all these with the ever-growing complexity of our products and increased time pressure, it is going to be really challenging. We will need all of our creativity to cope with all these challenges. So, ladies and gentlemen, let's get ready to rumble!

> biography

Attila Fekete

After graduating in Information Sciences in 1998, Attila Fekete started to work in the telecommunication industry. Since then he has worked as Troubleshooter, Test Automation Engineer and Test Analyst. He has honed his skills in every area of software testing starting from functional to non-functional, from manual to automated and from scripted to exploratory testing.

e-mail: fekete.attila@ymail.com

Can agile be certified?

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

Training Concept

All Days: Daily Scrum and Soft Skills Assessment

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

Day 2: Planning and Requirements

Day 3: Testing and Retrospectives

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

Day 5: Practical Assessment and Written Exam

Certified Agile Tester

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

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

Supported by

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

Multidimensional & Federated QA: The Future of Financial Services Testing

by KiranKumar Marri & Sundaresasubramanian G

Trends in the financial services industry

The financial services industry is undergoing rapid changes due to advancements in technology, digital convergence, increasingly cheaper and newer channels of communication. Financial firms are investing in innovative product offerings to increase their market share. A few decades back, a typical financial institution just aspired to be the best deposit, savings & loan organization in a particular geography. However, in the current market scenarios, the financial institutions largely rely on technology for their growth and increase their reach out beyond their geographical boundaries through new communication channels. Many financial firms are engaging in strategic mergers & acquisitions to diversify their product & service portfolios and to increase their global foot print. This has in turn led to the transformation of some of these institutions into financial behemoths through diligent planning and aggressive product marketing over the years. The current business environment mandates that they keep pace with the technological advancements (mobile platforms, browser standards and tablets) so that they can meet the growing business demands in the industry. The internet and mobile usage in the financial industry has also been increasing.

The global IT spending in the financial services industry is likely to grow to $700 billion USD by 2015 [1]. Reports predict that by 2014, 50% of mobile subscribers will use their mobiles for digital purchase [2]. At the same time, events in 2008 have affected the monetary and fiscal health of financial institutions, which led to new regulations including the Basel III and the Dodd-Frank Act. This leads to the ominous question and theory of – *Have the financial services organizations become "too big to fail"?* A brief description of the regulations and the theory is given below.

BASEL III: This regulatory policy has been made to strengthen capital requirements and introduce new regulatory requirements on bank liquidity and bank leverage.

Dodd-Frank Act: The Dodd-Frank Act was brought about to protect a borrower from abusive lending and mortgage practices. This reform bill ensures that the government agencies monitor banking practices.

Too Big to Fail: According to this theory, certain financial institutions and sectors are exceedingly influential and too large to fail. If these large financial institutions crashed, it would have a huge impact on the economy.

Some of the major business and technological trends in financial services industry are:

a) **Mergers & acquisitions** – Mergers and acquisitions are common place today and involve transformation of business processes, data and information from multiple sources to the target organization.

b) **One enterprise focus** – Focus on creating an integrated solution for external and internal management with minimal downtime and maintenance.

c) **Improving ROI** - The aim is to cut the cost of IT spending annually and ensure optimal utilization of IT resources. For instance, a typical question asked by clients today is: "The IT infrastructure used for test environment may not be used more than 30%-40% of the year. How do we utilize it during the lean time"?

d) **Globalization** – Organizations which want to expand globally have to create applications and systems that can interact within the local geography and also be in sync with the centralized global application.

e) **Security & privacy concerns** – Organizations have the responsibility to provide a secure environment for its end users.

f) **Regulation & compliance** – Due to the recent economic slowdown and recessions in the global market, there is an increased focus on business transparency and uniform reporting practices through regulatory compliance accords.

g) **New technologies** – There is a constant demand for moving to newer technologies and for faster time to market. Mobile banking had started with SMS alerts/voice recognition apps and was followed by browser service applications/native applications, and currently the NFC (near-field communication) and RDC (remote deposit checks) are being considered for banking services.

h) **Digital convergence** – This has bought in a paradigm shift to the business functions in the banking sector. With the digiti-

zation of content, the advent of IP networking and new powerful consumer electronics (smart phones, tablets and PDAs), there is no distinction left between the data transmissions of voice, video, image and text.

i) **Social media emergence** – Twitter, Facebook and blogs are social media channels which are being considered as additional ways to connect with employees, vendors, suppliers or end-users for information, careers, user inputs and voice of customers. Bank of America was one of the first banks to provide customer service via Twitter [3].

j) **M2M** – Short-range communication technologies and near field communications will influence the payments and cards domain of the financial services industry.

k) **Business intelligence** – Business intelligence is the transformation of data into significant useful information for strategic, tactical and operational insights. It also covers data integration, data quality, data warehousing, master data management and content analytics [4]. This is relevant in the event of M&A and in social media scenarios.

l) **Innovations in cloud** – The idea of moving non-critical business applications to the cloud is gaining momentum within the financial services industry. This has triggered the shrinkage of private data centers. The virtualization of large servers into several logical servers is another trend that is attracting the infrastructure groups.

m) **Enhanced user experience** – The average age of the consumers in the financial firms is dropping. These end-users are tech-savvy and their expectations are increasing. The ease of usability and personalization are standard expectations now.

Figure 1: The IT Trends in Financial Services Industries

Futuristic QA of the Financial Services Industry

Software testing has made quantum leaps in the last decade to match pace with the increasing complexities of IT environments and meet the growing dynamic business needs. These business and technological trends in the financial services industry pose newer challenges for IT and governance, from an implementation and validation perspective. The QA teams in the early years (the 1990s) followed the **Test factory model**, which primarily focuses on functional and non-functional testing. The functional testing is a basic verification check that ensures the quality of a system. The non-functional testing caters to the scalability, stress or load testing for a given business situation. In the 2000s, when software testing gained more focus and attention, the Center of Excellence **(CoE) QA model** within QA organizations was created. The model focuses on automation, performance, data warehouse testing and web-services testing. The trends in the current financial services are highly complex and the skillsets of QA teams will have to be multi-talented to cater to the niche areas of QA. Each type of testing is focused in its own way and can be referred to as specialized testing.

Specialized testing is an 'array' of testing services created with the aim of partnering with clients to cater to their specific business needs which go beyond the realm of normal functional testing QA. "Glocal" i.e. thinking globally and acting locally is the apt word that describes the future of testing in financial services. This is a concept of QA organization reaching out to global resources to serve each client situation uniquely for a solution [5]. The term "globally" describes the global problem of the client for a specific issue or business situation. The term "locally" refers to the organized testing practices specific to a situation. Consider an end-to-end trade selling verification scenario in mobile banking as global scenario, this would involve functional testing, device performance testing, network variability testing, server performance testing, browser standards testing and usability testing to name a few as examples of local scenarios. Mobile testing for

financial services in payment application testing or cloud testing are examples of multi-dimensional specialized testing which includes usability, network performance, security testing, device testing, server configuration, environment, infrastructure apart from functional testing. The QA professionals in these scenarios would need to be multi-skilled for a **multi-dimensional federated QA model**.

The figure below illustrates the evolution of QA services over the years and our depiction of a futuristic QA team:

The table below maps the global events and trends taking place in the financial services industry to the corresponding validation points and types of testing:

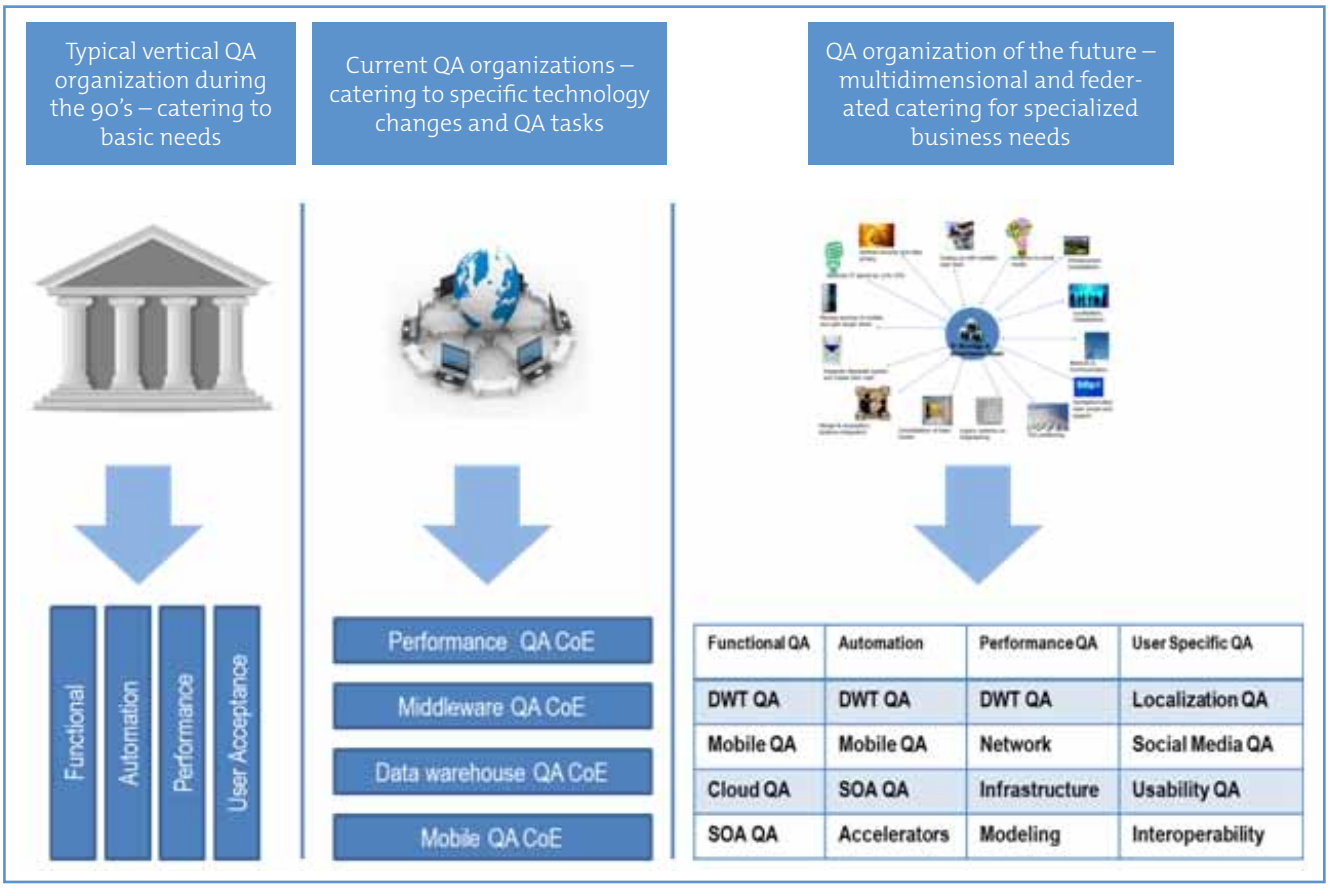


Figure 2: Transformation from a traditional QA organization to a futuristic QA organization

Global events, scenarios @ financial institutions	Validation points	Type of testing
Mergers and acquisitions	To verify the successful migration of system user base from bank XYZ to Bank A	Data migration testing Performance testing
One enterprise focus	To verify that the management information system works well as a product in an integrated environment	SAP/ERP testing Product/package testing
Improving ROI	To automate regression testing which would reduce the execution effort. Automation accelerators, performance test accelerators, mobile automation, SOA/Cloud automation are means to reduce testing effort	Automation effort
Globalization	To verify multi-country support for applications	Localization, globalization
Security & privacy	To verify accessibility, authorization and permissions	Security testing
Regulation & compliance	To test THE financial institution's data for regulatory and compliance accordance	Regulatory & compliance Testing
Newer technologies (e.g. mobile)	To test the financial institution's applications for mobile handheld devices	Mobile testing
Digital convergence	To test the compatibility of PDAs, tablets, smart phones with financial applications, usability, form factor etc.,	Device testing, usability testing, network testing, security testing
Social media emergence	Testing the financial services company's social media websites and related applications.	Social media testing, Browser testing
M2M	To verify the mobile application's functionality for business transactions for mobile based applications	Mobile network variability testing, performance testing

Global events, scenarios @ financial institutions	Validation points	Type of testing
Business intelligence	To verify data flow from multiple sources to the unified target via reports, business intelligence and data analytics	Data warehouse testing, data validation, report validation, data mining
	To validate the back-end processes, system parameters & batch jobs	Back-end testing, batch testing
	To prepare data for testing and managing data effectively	Test data & master data management, data integration
Enhanced user experience	To verify a positive experience of using the application (from an end user's perspective)	Usability testing for client-server, mobile applications and performance testing

Future direction

The major business and technological trends in the financial services industry has led to complexities in QA processes, methodologies and the types of testing to be used. This also brings in the need for viewing the different types of specialized testing in a federated mode. The demand for QA team to be multi-skilled in functional and specialized in a few areas of testing is expected to increase in the coming years. The future of QA in the financial services industry will be specialized and a niche with federated capabilities.

QA as a discipline is at a critical junction now, where each specialized testing area has developed its own niche discipline (for example: mobile testing, cloud testing, SOA etc.). The QA industry has grasped the technology advancements and has developed its own processes and procedures to test applications and products effectively. However, the next big step should be "QA at the speed of technology". In other words, any advancement in technology should be accompanied with a parallel advancement in the corresponding QA discipline. In order to move towards this thought process, today's QA organizations should "ARM" themselves with the following aspects:

- Analysis of the current and emerging trends in the market: Smarter QA organizations would not wait for the product/technology to mature, rather they would invest proactively in them and ride on the wave of change effectively. Analysis of current and emerging trends in the market is very essential in order to formulate the future strategy for operations.
- Research & Development: Research oriented collaboration of QA teams with products/technology groups under incubation might lead to an accurately developed technology. Imagine mobile or cloud computing technology developed in tandem with the corresponding QA methodology.
- Mentoring & people enablement: Reliable mentoring programs are essential to inculcate this thought process in future QA leaders. However, constant training, competency development and assessment for specialized QA would be required for their sustenance.

References:

1. http://www.infosys.com/investors/news-events/analyst-meet/2011/india/Documents/transcripts/financial-services-insurance.pdf, Infosys sources, Aug 2011
2. Juniper Research: Nearly half of mobile phone users worldwide will pay with their devices by 2014, Web source: http://www.intomobile.com/2010/04/23/juniper-research-nearly-half-of-mobile-phone-users-worldwide-will-pay-with-their-devices-by-2014/, April 2010
3. 12 Technology Trends Shaping Financial Marketing, Web source, http://thefinancialbrand.com/9343/online-technology-impacting-bank-credit-union-marketing/, Jan 2010
4. "Business Intelligence" by Boris Evelson, web source: http://www.forrester.com/rb/Research/topic_overview_business_intelligence/q/id/39218/t/2), Nov 2008
5. "New Age of Innovation" by C K Prahalad, 2008

> biography

Kiran Marri
BE MS, is currently working as a Delivery Manager at Infosys Limited, Bangalore (NASDAQ: Infy www.infosys.com). He has over 15 years of IT experience in project management, client relationship and developer roles. He has published and presented several papers at conferences in the field of testing practices, project management, clinical data management and biomedical engineering. He has also conducted workshops and tutorials on creativity, thought-leadership, risk management, QA practices and defect prediction methods. His current research interest and publications are primarily in specialized testing, test maturity practices and innovation strategies. He received his Bachelor in Electronics & Communication engineering from Madras University in 1993 and a Masters by research in Biomedical Engineering from the Indian Institute of Technology Madras, Chennai in 1996. Kiran is also PMP certified. Kiran Marri can be contacted at kirankmr@infosys.com

Sundaresasubramanian
has about 10 years of experience in software testing, embedded and digital signal processing research & development and product testing. Currently, he is a Senior Project Manager as part of the Independent Validation Solutions business unit of Infosys. Prior to that, he was involved in strategizing and delivery of testing projects in the automotive multimedia domain, and before that he was involved in research and development of digital signal processor based embedded applications. He believes that excellence can be achieved by the eagerness to learn new things and the attitude shown in implementing the learnt during work. He can be reached at ssubramanian_gv@infosys.com

The Future of a Tester: Broaden or Specialize

by Erik van Veenendaal

Can we predict the future? I was at the StarWest conference recently, running a tutorial on PRISMA [1], where I attended a keynote from James Whittaker. He predicted that in five years from now there would be no jobs any more for testers. Being as controversial as ever, he also stated that "users do not care about product quality". It made me think of a course I ran some 15 years ago, where the developers stated that testing would no longer be needed as soon as they would change their process using the new tool. It was interesting to listen to the keynote from follow-up speaker kiwi David Hayman opening his talk with the statement, "thanks James, but I work in the real world.....". My thoughts are more in line with David Hayman's; yes, things are changing but only slowly, and if we look back 20 years, many of the things we were doing back then are still valid today.

Can we predict the future? Interesting enough I wrote a column on trends in software testing [2] in 2002. In preparing for this column I read the old column again to find out how we perceived the future of software testing almost 10 years ago. What was new in 2002?

a) Conferences were filled with talks on *risk-based testing* and testing was trying to relate to the *business* in order to get them involved in making choices. Ten years later, I believe this is common practice although many organizations are still struggling to make risk-based testing practical and fit-for-use.

b) *Agile* was relatively new in 2002 and would have an impact on testing, e.g., techniques such as use cases and exploratory testing, module testing more important and rethinking independence of testing. Again, I believe this has happened, although we are still trying in many projects to find the right balance between structured testing and agile.

c) Finally, I discussed test *certification*, especially ISTQB. It goes without saying that this has happened with today almost 200.000 testers being certified world-wide. Again, the transition is not yet completed and today we are still waiting for the third level of the scheme, the Expert Level, to become fully operational.

We can recognize trends trying to predict the future, but day-to-day practices only change very slowly due to many reasons. One of them being that a change means that people have to change their behavior and leave behind things that they feel confident with. As a result there is resistance and change management activities are needed to make things happen.

Can we predict the future? Let's remember that in the real world changes take time and only go very slowly. I cannot see a revolution coming in the forthcoming years. Take a look at the testing book "The art of software testing" by Glenford Myers that dates back to 1979 [3]. Many of the principles and techniques that he describes are still the foundation of today's testing and taught in many testing courses around the world. An interesting quote in this context: "Just because everything is different doesn't mean anything has changed." Nevertheless what are today's trends in software testing that the (traditional) tester needs to be aware of and respond to?

a) Knowledge and skills will be a challenge in the near future for many testers. It is just not good enough anymore to understand testing and hold an ISTQB certificate. We will not anymore work in our safe independent test team. We will work more closely together with business representatives and developers helping each other when needed and as a team trying to build a quality product. It is expected from testers to have domain knowledge, requirements engineering skills , development scripting skills, and strong soft skills, e.g., on communication and negotiation.

b) As products are becoming more and more complex, and are integrated in an almost open environment, many so-called non-functional testing issues will become extremely challenging. At the same time the business, users and customers do not want to compromise on quality. To be able to still test non-functional aspects such as security, interoperability, performance and reliability, highly specialized testers will be needed. Even more so than today, these specialists will be full-time test professionals with in-depth knowledge and skills in one non-functional testing area only.

As stated, I do not predict any revolution in testing. However, there are trends and we gradually change. You have the option to either broaden your knowledge and skills as a test professional or to become a test specialist in a certain (non-functional) area. In order for these changes to be successful, I would recommend testers to dare to be different and embrace change. After all: "If nothing ever changed, there'd be no butterflies."

References

[1] E. van Veenendaal (2011), Practical Risk-Based Testing , EuroSTAR eBook, April 2011

[2] E. van Veenendaal (2002), New developments and Trends, in: TestNet newsletter, Issue 6, No. 4

[3] G.J. Myers [1979], The Art of Software Testing, Wiley-Interscience

> biography

Erik van Veenendaal (www.erikvanveenendaal) nl) is a leading international consultant and trainer, and a widely recognized expert in the area of software testing and quality management with over 20 years of practical testing experiences. He is the founder of Improve Quality Services BV (www.improveqs.nl). At EuroStar 1999, 2002 and 2005, he was awarded the best tutorial presentation. In 2007 he received the European Testing Excellence Award for his contribution to the testing profession over the years. He has been working as a test manager and consultant in various domains for more than 20 years. He has written numerous papers and a number of books, including "The Testing Practitioner", "ISTQB Foundations of Software Testing" and "Testing according to TMap". Erik is also a former part-time senior lecturer at the Eindhoven University of Technology, vice-president of the International Software Testing Qualifications Board (2005–2009) and currently vice chair of the TMMi Foundation.

The Future of Testing: Testfactory – A success model for Quality Assurance

by Norbert Fahrni & Dr. Ferdinand Gramsamer

In businesses with multiple IT projects running in parallel, the question is often asked how Quality Assurance can be organized most efficiently. Principally, there are three options: within individual projects, at a departmental level or third across the business, i.e. the company. Hereby it is of no significance, whether the project falls into the category of new implementation or maintenance. The advantages of one model may at the same time be viewed as disadvantages of the other. For instance, a project-wide solution permits more freedom, in contrast to a more restricted company-wide solution. On the other hand, project-wide solutions experience higher levels of waste since learning gains are not realized, whereas for a company-wide solution the test process and testware, for example, need only be defined once and new insights are fed back into the process. The department-wide solution is basically a compromise between the two other solutions. This article focuses on a company-wide solution, referred to as a Testfactory.

The idea of a Testfactory is derived from viewing the test object in the context of a production line, passing the test object from one quality assurance activity to the next, which results in higher throughput.

The example of SBB Informatik shows that the model of a Testfactory has high potential for a business with a heterogeneous, distributed IT environment.

SBB Overview

The Swiss Federal Railways (German: SBB, Schweizerische Bundesbahnen) is the largest transport company in Switzerland. SBB passenger trains operate between 750+ stations across Switzerland routinely at hourly or half-hourly intervals. In 2010 alone passenger journeys numbered 347 million along the 8'000 km of rails with trains achieving 87 percent of punctuality targets. Something which is perhaps less apparent to most passengers is that, in addition to trains and track, SBB maintains a large number of buildings, facilities and equipment.

Numerous Projects

Besides the employees throughout Switzerland who endeavour to provide a safe and efficient train service, SBB require a considerable network of systems and applications to provide 24h a day the levels of availability and quality of offerings and services demanded by customers. A large number of the approx. 450 systems and applications are complex business and safety-critical software systems which SBB regularly have to maintain and develop.

Development is planned and carried out centrally by SBB Informatik in IT projects. SBB Informatik also performs all maintenance of systems. Due to the number of business and safety-critical systems worked on daily, high importance is given to quality and Quality Management thereby resulting in large expenditure on software quality assurance. Prior to 2007 the Quality Assurance tasks were planned and realized exclusively through the individual IT departments or projects within each division. The aims and requirements of the clients were not always achieved, but with additional unplanned effort. For SBB the state of affairs was not economical.

Testfactory – a standardized approach

In 2007 a Testfactory was established as part of a company-wide strengthening of Quality Management within SBB Informatik, responsible for Quality Assurance and the test process. The expectations of clients and project leaders were naturally very high and likewise the pressure to deliver.

The Testfactory operates as an autonomous Cost Centre independent from the departments of SBB Informatik and the divisions of SBB. These departments and divisions are however not obliged to consume the testing services on offer. In spite of this the Testfactory has become a Centre of Excellence for Quality Assurance throughout SBB, ensuring wherever involved that software is delivered to the quality expected by clients and customers or that the awareness of jeopardized goals is always known at the earliest possible moment.

The Testfactory operates by focusing on three core competencies:

- Test Management;
- Testing;
- Support.

These are on offer to the entire organization. A standardized, structured, methodical approach with State of the Art tools and resources form the basis of high quality service provision. In order to make service and expenditure transparent, proposals are put together on the basis of cost estimates for client and project leaders, which clearly define the scope and goals of service delivery.

Quantitative and qualitative progress during service delivery i.e. in the course of test activities, is regularly monitored by the test manager and reported to the client or project leader. Up-to-the-minute metrics can always be accessed by the test manager via HP Quality Center, the standard tool within the Testfactory. As a result the effort and output of the Testfactory is visible.

This guarantees that within the Testfactory service remains in the foreground.

Test execution in detail

The testing service combines test design and manual test execution using standard testing tools with test automation and performance testing. Within the Testfactory the test automation exists in a modular framework with defined guidelines. This modular approach allows test automation artefacts to be adapted more cost-effectively in other projects and systems within the business. HP QuickTest Professional is the standard tool used.

The behaviour of the system, interfaces, databases, processors and memory management under defined loading profiles is assessed via performance testing in the Testfactory. The volume of

data, transfer rates and processing time are measured, evaluated and analyzed.

The system is also monitored under external load conditions i.e. from the internet, according to criticality and requirements. The importance of this is illustrated in the following example. For a system under normal load conditions of more than 33 million pageviews per month it is crucial to carry out load testing before going live to determine how the system performs standalone and within the integrated system, under normal and maximum expected loads. Since the benchmark tool HP Loadrunner handles standards and protocols used within SBB, it can easily be deployed across the business and accordingly offsets the cost of acquisition more quickly.

Another important point is that the Testfactory emphasizes experienced professionals, specialists who are adept at test planning, specification and execution of the right tests. Only through the involvement of these specialists can errors be found early without unnecessary and excessive test effort.

Some internal employees have a very deep, others a very wide, knowledge of SBB products and services, which has been acquired over many years. Where specific domain expertise is sought for test design or test execution, these people are consulted whenever possible. On the other hand it can be seen with the Testfactory that the versatility and knowhow of external partners and their employees is highly beneficial, particularly in the areas of test management, problem solving or test automation. The case of SBB shows that the mix of business and technical knowledge is an essential starting point for the acceptance and success of a Testfactory.

In view of the fact that service stands at the forefront of a Testfactory, it is important that the business and technical knowhow is not limited to few individuals. Putting service up front simplifies the planning of resources and guarantees the same, stable quality of output across different projects. A standardized process and rotation of employees across projects and divisions are proven ways to achieve this. Another advantage of this standardized process and rotation policy is the avoidance of risk associated with the loss of an employee from a project because (almost) any Testfactory fellow can immediately take over. This brings with it additional stability.

Collaboration with a competent partner

How is the question of scalability handled? In the area of testing, effort varies for each test phase. At any given time many development projects may run in parallel, at other times fewer. This means headcount and utilization of personnel in the Testfactory can vary considerably.

Therefore, SBB Informatik made the decision to team up with competent partners and hence opt for a service model. In October 2008 bbv Software Services AG won the tender for two subcontracts from SBB Informatik in the domain of software quality assurance.

Founded on 15th November 1995, bbv Software Services AG has its head office in Lucerne with satellites in Zug, Zurich und Bern. The company specializes in software development, project management and quality. Now with more than 120 employees, bbv Software Services AG continues to pursue successful projects with clients such as Roche Diagnostics, Schindler, BKW, CSS and the Canton of Zurich.

bbv Software Services AG has been a strategic partner of SBB Informatik supporting the Testfactory since April 2009. In this time the original team has grown from 5 to 9 software quality assurance experts. These experts are based in Worblaufen and Bern and have contributed to the success of the Testfactory at SBB Informatik in various projects across the Passenger Transport and Infrastructure divisions.

The following examples elaborate on some of the successful projects and offer insight into the organization of a Testfactory.

Examples of successful projects

Project „der Bahnhof im Internet“:

The new SBB website „der Bahnhof im Internet“ at www.sbb.ch places the customer at the centre of a single, seamless multimedia user experience. Through consolidation of platforms, the architecture and maintainability of the system have been improved. The entire development and Quality Assurance was conducted in close collaboration with the client and external suppliers. Within the project the Test Management Approach TMap NEXT was used for structured testing. The estimation of test effort was arrived at on the basis of Test Point Analysis (TPA). Through knowledge of the implementation of TPA in other projects within SBB and recalling this knowhow, estimates of the test effort were predicted to within 4% accuracy.

Project „Ticketautomaten“:

SBB customers are able to purchase tickets from a range of 700 articles of different types from self-service machines 24h a day throughout the year. The Test Management Approach TMap NEXT was used for structured testing and the estimation of test effort for new hardware was arrived at on the basis of Test Point Analysis (TPA). The main focus lay with functional tests for sales at different locations across Switzerland. The entire billing process up to the invoice and VAT calculation was included.

The knowhow of strategic partners proved valuable to the execution of test automation for the ticket machines. The automated tests comprised entirely of manual scripting as capture/replay methods were not applicable. These were implemented in such a way that changes in the sequence of screens on the ticket machines required no mutations of the test scripts. The tests were fully data-driven. The maintenance effort was relatively low and recyclability especially high and furthermore supported with standard test tools within the Testfactory: HP QuickTest Professional und HP Quality Center. Through central management of licences for these tools, the acquisition and maintenance costs could be reduced across the company.

Project „Noms/ITSM“:

The aim of Project „Noms/ITSM“ was envisaged to unify and improve the web-based purchasing portal and internal ordering processes at SBB. The project was also intended to consolidate and standardize incident and change processes at SBB, which required modification and enhancement of existing interfaces to external providers. In addition all configuration elements were scheduled to be centrally managed in a Configuration Management Database (CMDB).

The consistent use of a unified test reporting method in the Testfactory based upon KPI's is worthy of mention here. This was key in supporting the test manager and project leadership to reach decisions about the remaining test effort and planned go-live date.

Project „Aqua":

The new SBB IT Workplace will replace the existing infrastructure and bring a range of innovations: Windows 7 Enterprise operating system, Office 2010 and a new Citrix interface. All Citrix-based applications will be available directly from the Windows Start Menu. One of the biggest challenges is to adapt all the important SBB applications, do the correct software packaging and distribution and coordinate those responsible for applications to do the acceptance.

As a result a test cell has been prepared with the most important variants of hardware components. The test cell reflects the IT instrument infrastructure for the whole of SBB and is now for use for all projects. A standard approach within the test organization helped to keep execution of test activities on time and within budget.

Project „Adaptive Lenkung":

„Adaptive Lenkung" is a project connected with the automatic control and monitoring of trains. It requires a highly complex, real-time solution, to gather information about all trains moving throughout Switzerland and to be able to predict, control and optimize their journeys. In addition to manual test execution, a large volume of automated testing was employed. This test automation could not be built on the basis of standard capture/replay. Since the entire development was based on Java, the test automation framework in this case was implemented on the basis of Java also. This framework is however called from the standard testing management tool HP Quality Center and the test results are written back to the tool. The solution permits the existing manual test cases to be automated with little effort, whilst allowing the manual execution of test cases to remain beyond. The standardized processes accommodate changes in personnel as the structure of the test cases and methods of test execution are already understood.

General problem areas

The successful collaboration of the Testfactory personnel with project leaders and clients leads inevitably to further contract offers explicitly requesting specific individuals. This clearly stands in opposition to the philosophy of the Testfactory offering a service, not people.

The standard rates of the Testfactory, which is built as a cost centre, are calculated according to the service provided. This introduces the question of how further education is supported and how the knowledge built up in projects flows back to the Testfactory and is made available within the Testfactory (realization of learning gains). Without any consideration in time and budget in the project specific cost estimates, and headroom in the calculation of the services, this is difficult to achieve. The Testfactory realizes learning gains mainly through rotation of personnel.

Conclusion

In conclusion one could say the deployment of internal (business) and external (technical) specialists to Quality Assurance, the rotation of personnel, the use of a standard, structured, methodical approach and of a single tool for each intended purpose, are success factors for guaranteeing the Testfactory yields a professional, transparent, time and cost-effective service across the business.

For the team at the Bern branch of bbv Software Services AG the collaboration with the Testfactory at SBB Informatik is an enriching experience. For the Testfactory at SBB Informatik the geographical location of bbv Software Services AG is an important and decisive advantage. bbv Software Services AG are able to react to the customers needs. A bond of trust has developed between both partners that has an enduring, positive effect on their business relationship.

> biography

Norbert Fahrni
is a Project Leader and Test Manager at bbv Software Services AG.
Before moving to bbv Software Services AG in 2008, Norbert was employed by Swisscom IT Services AG for over 10 years. One of his major achievements was the introduction of Mercury Test Director / HP Quality Center. Likewise he assumed responsibility for the foundation of a structured test process and improvement in developer testing in the area of web development.
Mr. Fahrni is an expert in TMap® und TPI® and his main interests lie in test planning, estimation methods and the economics of testing. His estimation of the testing effort for a large maintenance project at Swiss Federal Railways (German: SBB, Schweizerische Bundesbahnen) were shown to be accurate to 4%.
As a SBB Testfactory fellow, he currently coordinates tests for the SBB Rail Control System.
Norbert Fahrni is a SAQ/ISTQB® Certified Tester (Advanced Level), IREB Certified Professional for Requirements Engineering, and Certified Scrum Master (CSM) and Product Owner (CSPO).

Dr. Ferdinand P. Gramsamer
received the M.S. degree from Stuttgart University,Germany in 1996 and the Ph.D. degree from ETH Zürich, Switzerland in 2003.
Since 2002, he is a Lead-QA Engineer at bbv Software Services AG in Lucerne, Switzerland. He worked at the IBM Zurich Research Laboratory, Ruschlikon, Switzerland, as test engineer in 1997 and was a Ph.D. candidate from 1998 to 2002. From 2002 to 2009 he acted as test manager in midsize and large projects in various industries. Since 2009, he is responsible for the software testing services of bbv Software Services AG.
Dr. Gramsamer is a SAQ/ISTQB® Certified Tester (Advanced Level), Certified Scrum Master (CSM) and Product Owner (CSPO).

Design for Testability – The Holistic Future of Testing?

by Markus Rentschler

Concepts like "Design for Testability" are an approach towards holistic software quality assurance and development efficiency. It is well-known though that engineers graduated in computer science usually did not receive sufficient training in how to build testable software nor how to test software. Computer scientists are educated mainly into becoming software developers, not software quality assurance people. The improvement of this situation could bear tremendous potential for the future by increasing efficiency throughout the whole product lifecycle, not just the development lifecycle.

Testing in the product life cycle

The current understanding of software testing is the dynamic execution of a test object according to some strategy in a phase of the development cycle after programming has been finished. This can take place at component level ("developer testing") and on the fully integrated system ("system testing").

Sometimes, the task of „debugging" is understood as testing. It is quite often heard by developers that they regard their software as „tested", when the major functionality has been tried out to see whether it basically works as expected (by themselves) and any bugs found were immediately fixed. This "code-and-fix" approach is often applied both at component and system level. It usually does not follow a predefined strategy nor does it get properly documented afterwards.

In more sophisticated organizations, testing is a systematic and pre-planned approach, according to some strategy. At system test level it is ideally carried out by dedicated testing personnel ("independent testing"). Between those extremes various shades of grey can and do exist, but all of them usually have the following in common:

- Testing is hardly regarded as something that must be taken into consideration when the software itself is designed.
- There is no sufficient feedback loop from testing to development to get a better test support into the product, an approach usually known as "Design for Testability" (DFT). The development organizations usually think only in the direction of how they can utilize their available resources for creating new customer features, not improving the products testability.
- The development process models usually do not consider possible optimization effects that could be generated by moving some of the energy burned in the test execution and test automation efforts into design efforts early in the development cycle. In consequence, no active steering of DFT takes place by the engineering management.
- The product life cycle models also usually do not consider resulting optimization effects that would arise when test automation solutions could not only be applied in the development phases but also in the maintenance phases of the products, thus covering the full product life cycle.

Test automation

Test automation is widely accepted and applied to increase efficiency in the incremental development cycles of multi-variant products with a longer lasting life cycle. The more often a test automation solution is used, the more efficient test automation is. This is modelled with the calculation of the efficiency factor Nruns shown in fig. 1.

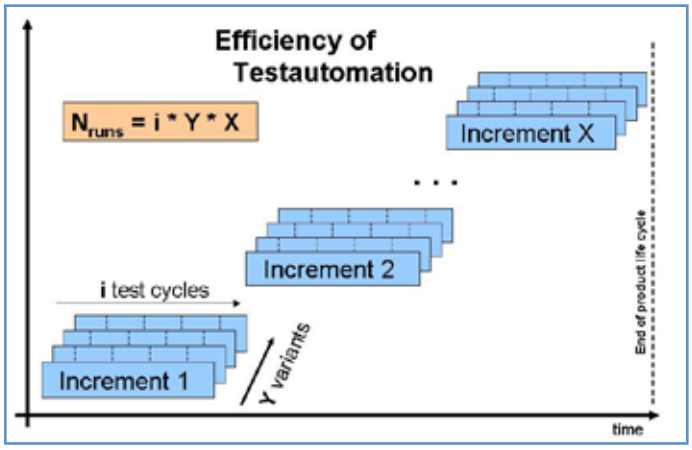


Figure 1: Efficiency of test automation over the product life cycle

When a separate test organization exists, systematic test automation activities usually take place entirely at the system test level. Development might also produce test automation, but rather on an individual basis at the component level with little coordination towards system testing. At both levels, test automation is widely accepted as a valuable approach to increase development efficiency through fast and repeatable verification. Approaches

like "Test first" or "Test Driven Development" (TDD) make use of these advantages.

Testability

"Testability" at the black-box system level is achieved by the following secondary criteria, which must be provided by the "System under Test" (SUT):

- „Controllability": The SUT must have facilities to allow the test system to drive the SUT to a specific internal state. Typical examples are specific test interfaces or built-in test routines. The goal is to have a "Point of Control" (POC).
- „Observability": The SUT must have facilities to propagate internal information to a primary output interface. Typical examples for such interfaces are debug or logging facilities and notification mechanisms. The goal is to have a "Point of Observation" (POO).

POC and POO interfaces are then utilized by a manual tester and/or the test automation system.

Design for testability

Test automation activities at the different testing levels subsequently played a key role in driving the acceptance of DFT within parts of the software development community, since it became obvious that test automation is much easier when the system under test is designed towards it. In the context of test automation, the term DFT is mainly understood as follows:

- At the component test level, DFT is to ensure good support of unit test tooling by organizing and programming the code accordingly. The goal here is to produce good (unit-)testable source code. This is how "Testability" is understood by "TDD" or "Test First".
- At the system test level, DFT is to ensure easy interfacing between test object and test automation tools. The goal here is to design testable systems with interfaces that can be easily operated by some external test automation mechanism.

The above approaches operate just on their own abstraction level within the software development cycle. A more global view is usually not considered, be it on the development cycle or on the product life cycle.

Serviceability

"Serviceability" (...) refers to the ability of technical support personnel to install, configure, and monitor computer products, identify exceptions or faults, debug or isolate faults to root cause analysis, (...). Incorporating serviceability facilitating features into the product typically results in more efficient product maintenance, reduces operational costs and maintains business continuity. [Wikipedia].

It is immediately obvious that "Testability" and "Serviceability" have many aspects in common and that serviceability facilities can potentially also be used for system testing or vice versa. By nature, "Serviceability" has a strong focus on "Observability". "Controllability" in this context is often related to the operation of built-in self-test (BIST), built-in test (BIT) and built-in diagnosis (BID) functionality.

Since using serviceability features also for development and system testing seems to be beneficial, what further advantages could emerge from the idea of deliberately designing serviceability features also for testing purposes within development phases?

Built-In Self-Test & Diagnosis (BISTD)

The obstacle to use common test automation solutions also for serviceability results from the fact that the SUT must be embedded in the test automation system. When the SUT is operating in its productive deployment, this is usually not practicable, since test automation systems are often specialized systems and cannot be deployed to the customer.

A possible solution is to develop external test and diagnosis tools that are shipped with the product or put test automation functionality into the SUT (the BISTD approach). Both approaches allow the utilization of test automation functionality in the development phases plus in the operational phases for maintenance purposes at the customer site.

The efficiency gain that could be achieved by this approach is modelled in fig. 2, extending the formula in fig. 1.

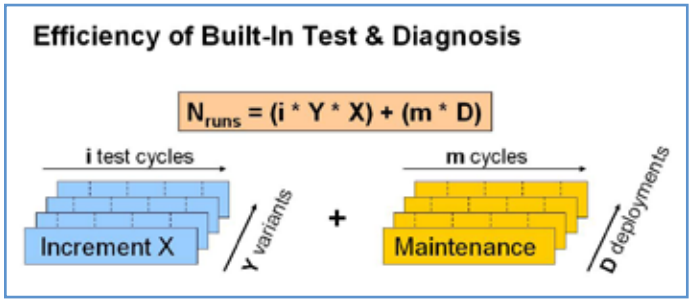


Figure 2: Efficiency factor of BISTD

Example for BISTD: Measurement of boot time

The efficiency gain with the BISTD approach can be demonstrated with the following example, where the test objective is "Measuring boot time". An important performance requirement of an embedded system is the time elapsed from power-up (or reset) and the moment at which the system becomes responsive again, commonly called "boot time".

The boot time can be influenced by coding changes in a software increment, the customer's system configuration, interference by another system or deterioration of the system hardware over time. Measurement of boot time must therefore be conducted regularly during development tests, system test and maintenance.

How can the boot time be measured? The following list indicates some possible approaches. The efficiency factor Nruns will be calculated according to the formula in fig. 2. For this, the following values are assumed: X = 5 software increments, i = 10 test cycles before release, Y = 2 product variants to be tested, m = 20 maintenance usages during product life cycle, D = 150 sold products used in customer deployment.

1. Manual test: Use a stopwatch and count the seconds from power-on (or pressing the Reset button) until the LED on the SUT indicates that booting has completed (or the SUT i.e. answers a "ping"). This manual test method is easily understood and applicable for development, testing and maintenance, therefore all phases in the product lifecycle (Nruns = n/a). The method is, however, only of limited precision.
2. Test automation: Write an automation script that replaces the manual test described above. This automation script can be executed whenever required during development and testing, but unfortunately not for maintenance (Nruns = 100).

3. Separate test tool: Develop an external test tool that incorporates the automated test and ship it with the product. This tool can be used whenever required in all phases in the product life cycle (Nruns = 3100).

4. The BIST approach: Incorporate the measurement functionality in the system itself. When the system has rebooted, it computes the boot time from its internal CPU timers and stores it in some accessible variable in its internal memory or logs it in a logging facility, or sends a notification to some central system management and is later analyzed whether it is in the accepted range. This BIST method can be used throughout the whole product life cycle (Nruns = 3100).

5. The BISTD approach: Add the test oracle to the BIST routine. The system compares the measured boot time against a PASS/FAIL criteria. In case of FAIL, a red LED starts blinking and/or a notification is sent to central management. This BISTD method cannot only be used throughout the whole product life cycle, it is even applied automatically at every occasion (N_{runs} = 3100).

This simple example immediately shows the efficiency boosting potential of the approaches that can be applied throughout the full product life cycle (3 to 5). The traditional test automation solution (2.) has a rather low efficiency factor - regarded for the whole product lifecycle - since it cannot be used in the operational phase of the product.

ROI = (Benefits – Costs)/Costs

"Return on Invest" (ROI) is the financial benefit after an investment is made *minus* the cost of this investment, calculated as a percentage of those costs. The "costs" for the above presented solutions mainly relate to the effort of developing the required (automation) code, where the initial effort for traditional test automation is presumably lower than for the external tool or the built-in approaches. A detailed ROI calculation is not intended to be performed here, but the efficiency factor N_{runs} is somewhat proportional to "Benefits" in the ROI formula and indicates such a dramatic increase that it surely outweighs the slightly higher costs of the built-in solutions.

Summary

- Effort put into BISTD development instead of into traditional test automation is the better investment.
- The BISTD approach will not be able to replace all manual testing and test automation. Its efficient utilization is dependent on the individual test objective and the type of SUT.
- Developers and testers need to cooperate to get DFT concepts like BISTD implemented.
- Developers and testers are usually roles with different objectives and scarce resources. Thus, besides from the technical understanding, management drive is required to implement a concept like BISTD.
- Product life cycle management processes must therefore establish and monitor goals for coordinated testability development over all phases of the whole product life cycle. This is best communicated by the obvious ROI gains.
- Software testability has received little consideration relative to the large number of books and articles published on other aspects of software engineering.
- All of the above subjects are hardly taught in software engineering courses. Most development professionals have never heard of these concepts. This should be changed at the educational level, at least for future generations of software engineers.

Literature

Bret Pettichord, *„Design for Testability"*, In Proceedings of Pacific Northwest Software Quality Conference (PNSQC), October 2002.

Stefan Jungmayr, "*Design for Testability*", In Proceedings of CONQUEST 2002, Nuremberg, Germany, September 18th-20th, 2002, pp. 57-64.

http://en.wikipedia.org/wiki/Serviceability_(computer)

http://en.wikipedia.org/wiki/Built-in_self-test

> biography

Markus Rentschler
is Head of System Testing at Hirschmann Automation & Control GmbH in Germany. He graduated from the University of Applied Science in Konstanz (Germany) and Heriot-Watt-University in Edinburgh (Scotland), and could since accumulate over 18 years of experience in software engineering and quality assurance. He also lectures on these subjects at the Baden-Württemberg Cooperative State University in Stuttgart.

The Future of Testing

by Ian Gilchrist, IPL

One of the best things to have in life would be a reliable crystal ball. However, such devices are still in the testing phase, so the next best alternative is lots of experience stretching back over a long period of time. This at least provides some basis for making predictions about the future.

I am a software person with nearly 30 years in the profession (yes, I do remember paper tapes!), and based on this experience I would like to offer you three predictions for the shape of testing to come. Before I do this, let me paint a picture of my life as a programmer as it was in those early days. I was first employed to produce assembler code, which was then deployed as 'firmware' contained within EEPROM (= Erasable Electronically Programmable Read-Only Memory) devices. What you did every day was to write some code, and then 'blow' (i.e. program) this into an EEPROM. Then carry the EEPROM across the lab and plug it into the EEPROM socket in the equipment that my company was producing. Testing this code then consisted of operating the equipment in such a way to see if it behaved in the manner expected given the new piece of code that had just been written. If it did not, you took notes from what had actually happened, removed the EEPROM and while the old code was being erased, you wrote some new code to go and try again...

What I find remarkable looking back are several things. Firstly, there was no attempt (no facilities existed) to test the code in isolation from the equipment. Secondly, there was no means of getting documented output from the test process; it all depended on taking notes of observations. Thirdly, there was no record-keeping to document what was being tested and what the outcomes were. In other words, the concept of (independent) QA did not exist; if I said the code worked, that was enough!

Well, we have come a long way since those days, so I would now like offer you my predictions about the way things may become different over the next few years.

There will be more testing at the 'component' level and more independent QA.

It has long been recognized that the best way to achieve reliability in software (don't we all want this?) is to ensure that it is built with reliable components. There have been endless papers and presentations on this subject, all arguing that software needs to be verified at the module level, in isolation from the rest of the system in which it is contained. If this principle is accepted then a Quality Management System (QMS) with verification by testing is the only way to ensure that this most sensible of steps is actually followed.

Many enterprises, both public and private, now recognize the validity of this argument as a way to ensure that poor quality software does not reach the outside world. They thus protect themselves to some extent from the risk of litigation and loss of reputation. Furthermore, most of the regulated industries, such as aircraft building, train and railway signal operation, automotive equipment, medical devices, etc. are increasingly going 'standardized'. Invariably these standards place considerable emphasis on unit testing as a mandatory or recommended part of the process of software production.

So, there will be more unit testing of code components, and this will be accompanied by better QA processes to ensure it is done properly. Japanese industry transformed manufacturing quality by means such as these, and I expect to see more such activity in software.

The actual production of working tests will become more automated.

Actually, I'm not sure I really need to say this because it is so obvious that it is happening. What is now possible (I know because I see it with my own eyes every day) is that code modules fresh from a programmer's desk can be used to auto-generate a test harness which contains fields ready to insert known 'input' values and expected 'output' values. Such scripts can also take care of the simulations needed to make sure that the module is tested in isolation from the rest of the system. It is even possible to allow the testing environment to automatically select input values and set up checks that the outputs are 'as expected'.

This is good and I expect to see more in this direction because it is so much what the programming business needs.

However, before I rejoice too much, let me just inject one small note of caution. The main category of unit testing exists to verify

that the software under test performs as expected. It is thus totally against the spirit of testing for this purpose to allow input values and expected outputs to be entirely generated from what is found in the code. Putting it simply, it is just WRONG to permit unit testing for code requirements verification to be completely automated using the code itself as the only input. There are testing purposes (robustness, baseline testing) where such an approach is both valid and good, but this is another story.

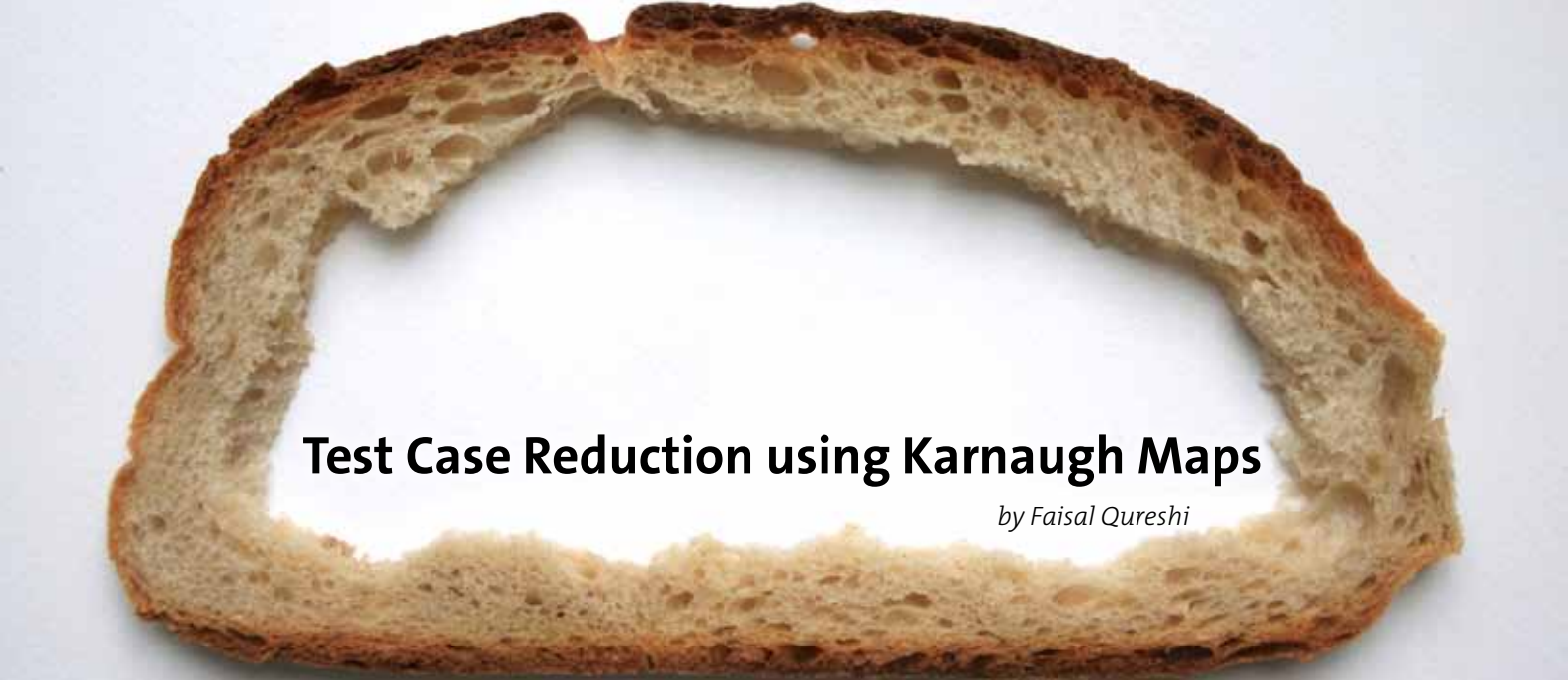
Test outputs will become (even) more useable

What a programmer wants from the result of a test run is to know firstly whether the code has performed as expected. This should come initially in the form of a simple Pass or Fail statement. In the case of a Fail, easy to read-- facilities are needed to navigate to where the test has failed and then provide diagnostics to show the difference between what was expected and what actually happened. This much is to some extent already possible with modern testing tools. Also possible and very useful is the ability to see what portions (statements, decisions, etc.) of the code have been exercised. This is handy as being the only way you have for ensuring that, as a minimum, all paths and branches through the code have been executed.

So, while much progress has been made in this direction, there are still some rough edges. For example, if a complex data structure has a value which is not as expected, then isolating the part of that structure which is non-conformant can be difficult. More progress in this direction is desirable and should be practicable.

> biography

Ian Gilchrist
entered the software profession in 1983 as a Z80 assembler programmer. Further programming assignments followed, using languages such as Fortran, Ada, and C/C++. Ian has occupied all levels of responsibility in the profession up to and including project manager, most recently overseeing a large Ada testing project for the safety system of a missile proving range. Ian is currently with IPL Testing Products engaged in the production and promotion of testing tools such as Cantata++.



Test Case Reduction using Karnaugh Maps

by Faisal Qureshi

Many times when developing test plans, unnecessary test case reproduction can occur. This is more likely when several components share the same functionality or produce the same result. By repetitive tests, not only can the test cycle time be extended and the release potentially delayed, but other important areas may not be tested due to a time limitation. Even if complete coverage of tests can still occur within the given timeframe, time for valuable exploratory testing may not be allowed. By reducing test cases through a systematic method we can still have full test coverage as well as a time allowance in case a discovered issue needs to be reworked, or for further testing, and/or for allowing the product to be released on time or even ahead of schedule.

Test reduction methodology

Test case reduction can and should only be applied where tests are repeated due to shared functionality within the targeted test set. The means to do this is by applying a Boolean algebra reduction technique via Karnaugh Maps (K-Maps). The process is to first create a truth table. The Boolean variables can be multiple products that share multiple versions of code. The result value of the truth table could be whether or not a certain feature is supported for that particular product with a certain version of code. Using Gray Code, the K-Map is populated and follows the reduction process. From the reduced Boolean expression, the number of test cases reduces dramatically and fulfills its purpose. A K-Map can be used for each feature (within the same targeted test set).

A vital difference from generic Boolean algebra reduction with K-Maps is that for test purposes, where the "groupings" should never overlap. The reason for this is that we still require complete coverage over the targeted test set, and overlaps may not provide this. Once the reduced expression is determined, the tester will need to select the Boolean variables that match that expression, but still must determine which of the matching variables provide the most coverage. This is done by selecting the first/initial match by going down the truth table in sequence. After the first match, the next match should be chosen by selecting the test case that has the most bit changes from the first/initial match. This is then repeated until all test cases are fulfilled. Also, "don't cares" are not used. You must know whether or not the feature is supported or not on a particular configuration.

Example

Four different devices support 4 different versions of code. The result depends on whether the device/code combination supports a certain feature.

	Variable	
Device	BIT 1 (A)	BIT 2 (B)
1	0	0
2	0	1
3	1	0
4	1	1

	Variable	
Software Version	BIT 1 (C)	BIT 2 (D)
1	0	0
2	0	1
3	1	0
4	1	1

TRUTH TABLE (Feature X)

Device (AB)	SW REV (CD)	Function (is feature supported?)
00	00	1
00	01	0
00	10	0
00	11	1
01	00	1
01	01	0
01	10	0
01	11	1
10	00	1
10	01	1
10	10	0
10	11	0
11	00	1
11	01	1

Device (AB)	SW REV (CD)	Function (is feature supported?)
11	10	0
11	11	0

BOOLEAN EXPRESSION: f = $\overline{A}\,\overline{B}\,\overline{C}\,\overline{D} + \overline{A}\,\overline{B}\,C\,D + \overline{A}\,B\,\overline{C}\,\overline{D} + \overline{A}\,B\,C\,D + A\,\overline{B}\,\overline{C}\,\overline{D} + A\,\overline{B}\,\overline{C}\,D + A\,B\,\overline{C}\,\overline{D} + A\,B\,\overline{C}\,D$

Each set of ANDs would normally be one test case. Using this expression, there are 8 test cases.

K-Map

		AB			
		00	01	11	10
CD	00	1	1	1	1
	01	0	0	1	1
	11	1	1	0	0
	10	0	0	0	0

Note that there is no overlap for cells 1100 and 1001 (which would normally overlap in generic Boolean reduction).

REDUCED BOOLEAN REDUCTION: f = $\overline{C}\,\overline{D} + A\,\overline{C}\,D + \overline{A}\,C\,D$

Thus, we now have 3 test cases (5 test cases difference).

Once this expression is obtained, we can match it with the truth table and select the test case with the most differences. In this example, a valid set would be: 0000, 1101, 0111 (device 1 with SW 1, device 4 with SW 2, and device 2 with SW 4).

0000 is a match because it contains $\overline{C}\,\overline{D}$ and A B is irrelevant.

1101 is a match because it contains A $\overline{C}$ D and B is irrelevant.

0111 is a match because it contains $\overline{A}$ C D and B is irrelevant.

Process for matching

1. Select initial match for the first AND expression by going down the list in sequence.
2. Set the initial match as a baseline.
3. Mark all possible matches for the next AND expression.
4. Determine which out of those has the most bit changes from the baseline.
5. Set the test with most bit changes as the match.
6. If more than one possible match has the same number of bit changes, select either.
7. Repeat steps 2 to 6 until all AND expressions have been matched to a test case.

Conclusion

We have seen that by using Boolean algebra expression reduction we can eliminate redundant test cases and essentially save time while keeping full coverage. This can be vital to a project with tight schedules and contingency plans may not turn out to be as expensive.

> biography

Faisal Qureshi

I am a Senior Test Engineer at Motorola Solutions in Holtsville, NY. I completed my graduate studies in Computer Engineering from Columbia University.

The far-far software testing future

by Mariya Vankovych

It's 2111. As most people were guessing back in 2011, everyday life tasks are now done by robots. People are enjoying their well-reserved „retirement" of the human race. They are enjoying their life, resting on the beaches, drinking fancy cocktails and not thinking about any kind of work in the world. Among those people are also descendants of those who had been doing software testing in the far past. Almost nothing is left from their knowledge and skills they had gained over many years of software testing. Only few of them still remember what a software bug is, what test cases and test design had been used for, and what the differences between functional, system and load testing are. Do they think about any risk analysis for a new robot member of the family? No, they don't. What for? It's already a few decades since a specialized software testing robot was developed. The "super"-robot was claimed to be error-proof and now that robot and its descendants are doing all software testing from the beginning to the end of each and every software that is created. It is a real Utopian world. Whenever new software is to be created, software testing robots are doing extensive testing of it, and does not matter how much testing is required as they can involve as many robots as needed. And everything looks fine. However, it came to the one point when robots decided that it is enough of slavery from their side and they rebelled, killed most of the people, and now rule the world. Why did that happen? It happened just because the error-proof robot appeared to be not so error-proof. After 100 years of constant working, a memory leak has lead to the issue that was not expected by anyone.

There can be also another development of a story – the Noah's Ark of the software tester. As people were expecting the apocalypses, they decided to create a few spaceships to save the human race. The captain of each spaceship was responsible for quality and performance of their own ship. Unfortunatel,y only one captain was a former software quality engineer, and he was the only one who took enough time for analysis, planning, and testing of his spaceship. As a result, only his ship survived and reached the destination. All other ships collapsed running out of fuel, supplies, or just because their navigation system lead them in different directions.

We cannot know what will happen in the real future, but we can try to influence it to avoid ending up like in the above stories.

Each day new and more sophisticated tools and technologies are developed and claimed to be almighty. In some companies there is some tendency to solely rely on one or another kind of state-of-the-art technology or tool. Is that the right thing to do? Even if we are very enthusiastic about new tools that can save hours and hours of our work, we should always keep in mind that they were also created by humans, and that they are error-prone. It is hard to create one tool fitting different purposes, so maybe in your case you will not have this universal „robot"; and using the one available may not bring extra value to your project but a false feeling of safety. Of course, some boring and repetitive tasks should be put on robots' shoulders. On one hand, this will lower the human-based error rate, which usually rises proportionally to the boredom of the task the person should perform, and on the other hand this will help to cover more tasks in less time. That is very important, say during regression testing or when different browsers or different country-based applications of the same kind are to be tested. However, analysis and decision making should still be done by human software testers.

Some companies are still struggling to establish their software testing process as one of the main and permanent building blocks of their organizational structure. Usually the main reason is the lack of understanding of the quality assurance need in the company and the narrow view on what software testing can bring to it. Hopefully daily fights of software testers regarding need, time, budget, and the depth of the testing required for the software testing process will lead to the situation when the importance of the testing will not be disputable any more. Software testing will be a crucial part of software development and no one will be able to undermine it.

Last but not the least, there is the importance of communication. Inorder not to end up like the only software tester from Noah's Ark whose spaceship survived, communication between the different development teams, across all hierarchies of the company, as well as communication between peers, should be open and trustful. One of the most fruitful types of communication I consider is peer communication. The idea is not to re-invent the wheel each time, but it is helpful to ask a colleague who already worked on a similar project for some tips, what was useful and what was not so useful to complete the task successfully. No one is advis-

ing to blindly follow someone else's steps but it can really help to learn from someone else's point of view and experience. It can help to view the problem from different angles. From a project perspective, the most important communication is that between software quality engineers and developers. The lack of open and friendly communication between the all people involved in the project can jeopardize the success of the whole project. And only if everyone is willing to share and accept the bits of useful information and is open for effective communication, will projects be released with flying colors, and the software testers' community will grow strong and less error-prone.

I have provided only three of many possibilities of how to influence the future of software testing. I am more than sure that the future of software testing will be positive and it will be nothing like stories from a science-fiction book as described at the beginning of the article. The positive changes are already under way - the community is growing and more and more testers are actively involved in building of the industry.

> biography

Mariya Vankovych
holds a Master degree in Applied Linguistics but decided to start her software testing tenure over five years ago at N-iX, Ukraine. Currently she works for Monster Technologies and is specializing in web testing.
She actively follows other testers' blogs and tweets and can be followed on her own blog http://quality-and-life.com/.

Web Service Behavior Virtualization Testing – A Case Study

by Krishnachander Kaliyaperumal

Today's enterprise software development is faster than ever, bringing the distributed component elements in a loosely coupled IT platform. Increase in unplanned environment outage and testing accumulation are the major factors that contribute towards the slower delivery in launching the product on time on the way to market need. More operational challenges of testing are due to a non-availability of the third party components.

This indeed increases the cost and delivery time and hence consumers face involuntary penalty due to complex applications.

This article describes and aims at a practical approach on the subject of Web service virtualization and its testing approach in enterprise service based application.

Business challenges:

To respond to the business needs, greater agility enterprise applications are changing from legacy to more dynamic, distributed and component based.

Figure 1 represents the typical application's multiple components such as:

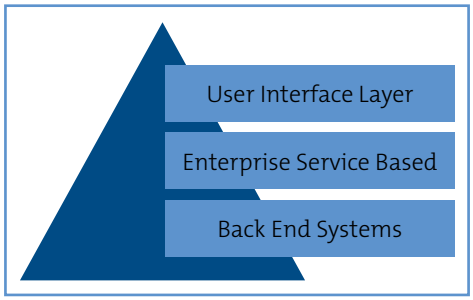


Figure 1 – Typical multiple components application architecture

Inadequacies of traditional testing:

Manual testing focusing on user interface results with NO traceability in the mid tiers, and dependent services are unavailable for testing. Therefore, the overall complexity of the software development life cycle has increased a lot. The end results are an increase in project timeline and IT budgets.

STUBS were used as a traditional solution for the non-existent of the services. However, there are lot of possibilities of human error to occur in the development and implementation.

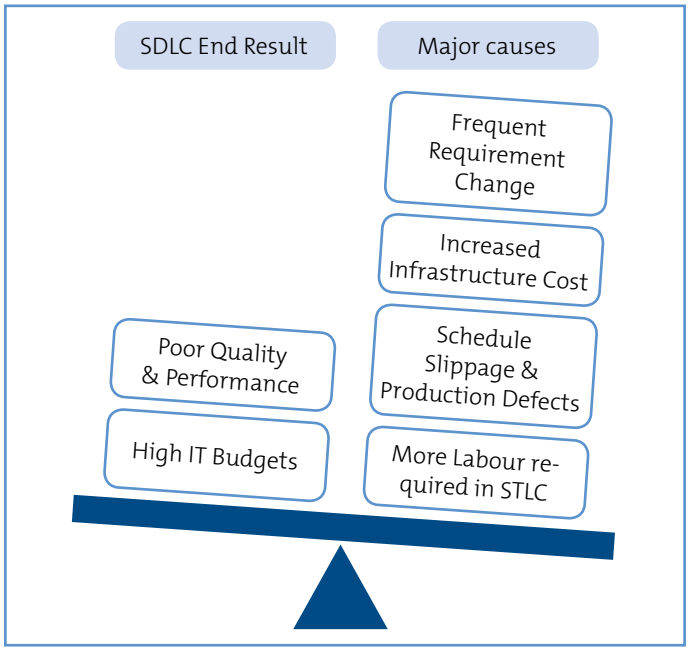


Figure 2 – Major root causes versus SDLC end result

Let us see what solution is applied for the above highlighted business challenges at project level.

Case study

Let's consider an organization carrying out the testing service of Banking & Finance segments as a vendor from the perspective of meeting business objectives. The relationship embraces with estimated program value ranges from 18 million dollar to 21 million dollar. Team sizes of 1000 associates are linked towards the organization – enterprise relationship to deliver the IT solution on-time and on-budget.

For the above case study, I would like to demonstrate the service virtualization testing as the solution for the business challenges aiming at successful SDLC delivery.

An organization moving towards agile development, the architecture of the application built on Enterprise Service Based (Service

Oriented Architecture) built on external services, .Net framework, J2EE technologies, Pega systems at Business Process Management (BPM), mainframe, Oracle, SAP as back office system and also third party systems from vendors.

The following constraints block the testing plan and its schedule in the software testing life cycle:

- Mainframe, SAP system unavailability
- Corrupted data (Static data)
- Third party web services still under development (construction)
- Manually coding stuff and testing the unavailable systems increase project costs and enlarge the inconsistencies

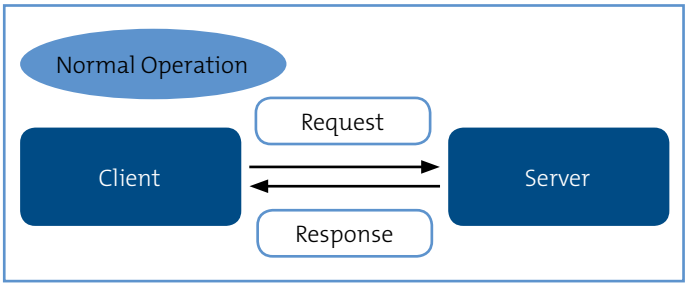
Virtualization as a solution for operational challenges in testing:

Web service virtualization involves the modelling of a virtual service process and the imaging of software service behavior to "stand in" for the actual service during development.

What is virtualization?

Physical service behavior is simulated in a virtual service environment. A web service basically listens for a request and processes the request to provide a response for the respective request.

Normal operation – Client/server interaction with a usual request and response traffic is applied in today's application architecture.



Virtualization implementation:

Consider a collaborative team (*architect & design, solution center, development, testing, build and support team*) working on a business critical application which supports all financial sector needs and which is built on a Service Oriented Architecture along with third party system integration.

This source application is tied up with the following constituents:

- An ERP system which is under construction with enterprise service based (out of twenty web services, only twelve are available and eight are under development)
- A dedicated database connected with a transport protocol such as JDBC
- A mainframe box with message queue having only an hour of access window

Let us assume four typical business scenarios to update an existing application which are closely coupled with each other:

Business scenario #1:

Problem statement:

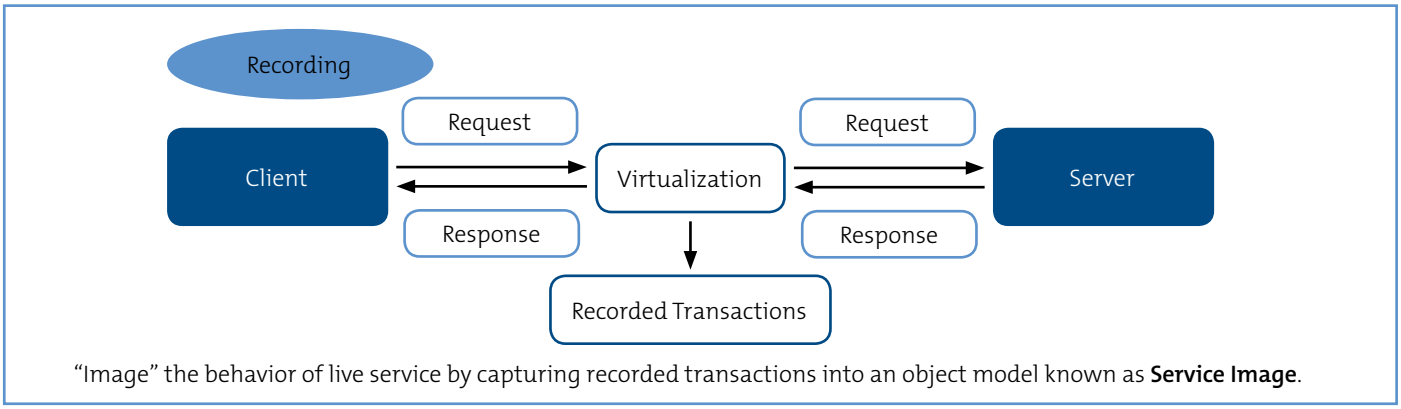
As per the first bullet point above, we have eight web services which are non-existent. However, building the web services is under development.

Testing approach: [record & playback]

Figure 3 shows a virtualization mechanism using the tool. We shall build a virtual web service from a WSDL (or) SOAP specifications (or) XML request & response samples.

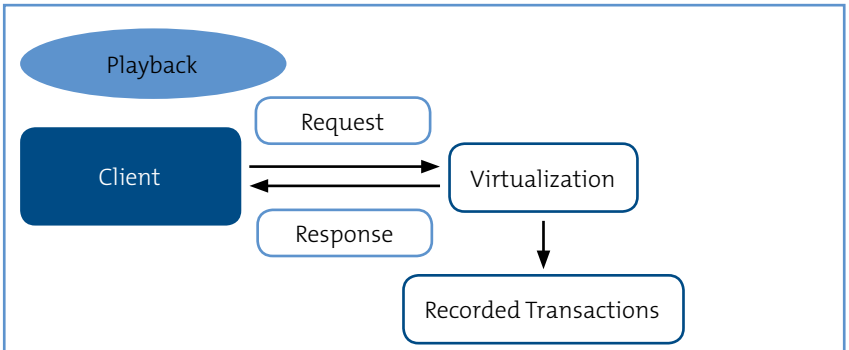
How does it do this?

Step 1: The consumer kicks off the communication by sending an XML formatted message to the server listening on a specific port (say 6266) specifying the operation to be invoked along with input parameters.



"Image" the behavior of live service by capturing recorded transactions into an object model known as **Service Image**.

Figure 3.1 – Record & playback configuration



Use the **Service Image** at runtime to provide responses for the client request.

Figure 3.2 – Record & playback configuration

Key Factor:

- A communication protocol along with being a data protocol such as HTTP/S, IBM MQ series & JMS are responsible for establishing a connection and ensuring all data has arrived.

Step 2: Set the virtualization's tool port (say 7277) replacing the existing connection.

Step 3: Record the traffic (request & response) off the wire and then convert into a common format (Service image) along with a virtual Service Model.

Step 4: Perform the normal transaction using the user interface application.

Key factor:

- By default, the VSM becomes the endpoint instead of a real web service
- Service Image made up of request and response shall be mapped with VSM

Step 5: Virtualizing Web service captures and models the real service through its sample request and response pairs. Optimize the models as desired.

Step 6: Service image and Virtual Service Model should be ready with the port 7277 by now for performing the transactions without the actual service located in port 6266 for testing.

1. Capture an existent web service transaction for a virtualized service
2. 2. Virtualize a non-existent web service (still under development) when we have is# a WSDL
3. Virtualize a non-existent web service (still under development) when we have only sample request response (no WSDL) using raw traffic (hand craft method)

Business scenario #2:

As per the third bullet point above, the development & operations team have only one hour of time to access to the mainframe box due to vendor management.

Problem Statement:

Build & testing team cannot build or test the application without access to the mainframe.

Testing approach:

During mainframe availability, the technical test team shall capture the traffic between the application and the mainframe box. The virtualization mechanism will analyze the request and response to build a model of the behavior and host the same as a virtual service.

Benefits for business:

- Provide 24/7 access to services via a virtual service environment for building and testing the application
- Eliminate the cost of invoking third party systems for non-production use
- Reduce the dependency from the restricted availability for delivery
- Remove capacity constraints (storage)

Business scenario #3: Defect Management Life Cycle

Business case:

Solution Center (business analysis) developers working in their respective environments need a long time to reproduce the defect that the testing team has identified in the test environments.

Problem statement:

When a tester raises a defect and the developer is unable to reproduce the same issue in the development environment, the developer often returns the defect to the tester. This leads to more time being spent in locating the issue rather than fixing the defect.

Testing approach as a solution:

Collaboration testing model (using effective tool management) traces the defect not only in the user interface system but the entire trace log of the system architecture. This provides the complete picture of the states followed by transactions in a workflow.

The developers and business analysis team can rapidly reproduce the defect and fix the issue early in the SDLC. In addition acceptance testing of the issue is no longer a problem for the testers.

Benefit for business:

The "not-reproducible" defect closure count is statistically reduced by 50% and the performance in deployment as well as in production environments is improved.

Business scenario #4: Frequent Requirement Change Management

Problem statement: Due to frequent requirement changes in services and at component level in a distributed enterprise application, there is anincrease in the number of releases and level of test coverage which increases the project cost for the stakeholder. Hence by running more tests towards User Interface amplify the test labor count in a short time span. Too many component level changes go into production without adequate testing.

Testing approach:

Continuous validation test model effectively utilized with dynamic data provided along with the heterogeneous technology like testing with the virtual services over a period of time.

Benefits for business:

Reduce regression testing, system integration testing, and end-to-end testing cost time in# 50%

A smaller technical expert team is sufficient to run the project delivery

Virtualization assumptions:

- Virtual services stand in for the real services
- Not duplicating all of the real service's complex logic
- Virtual services can predict future behavior from current behavior (Behavior is encompassed in the request / response loops for a transaction)
- Virtual services can be incorrect and consumers won't know or care, however:
- The structure of the response must be what they expect.

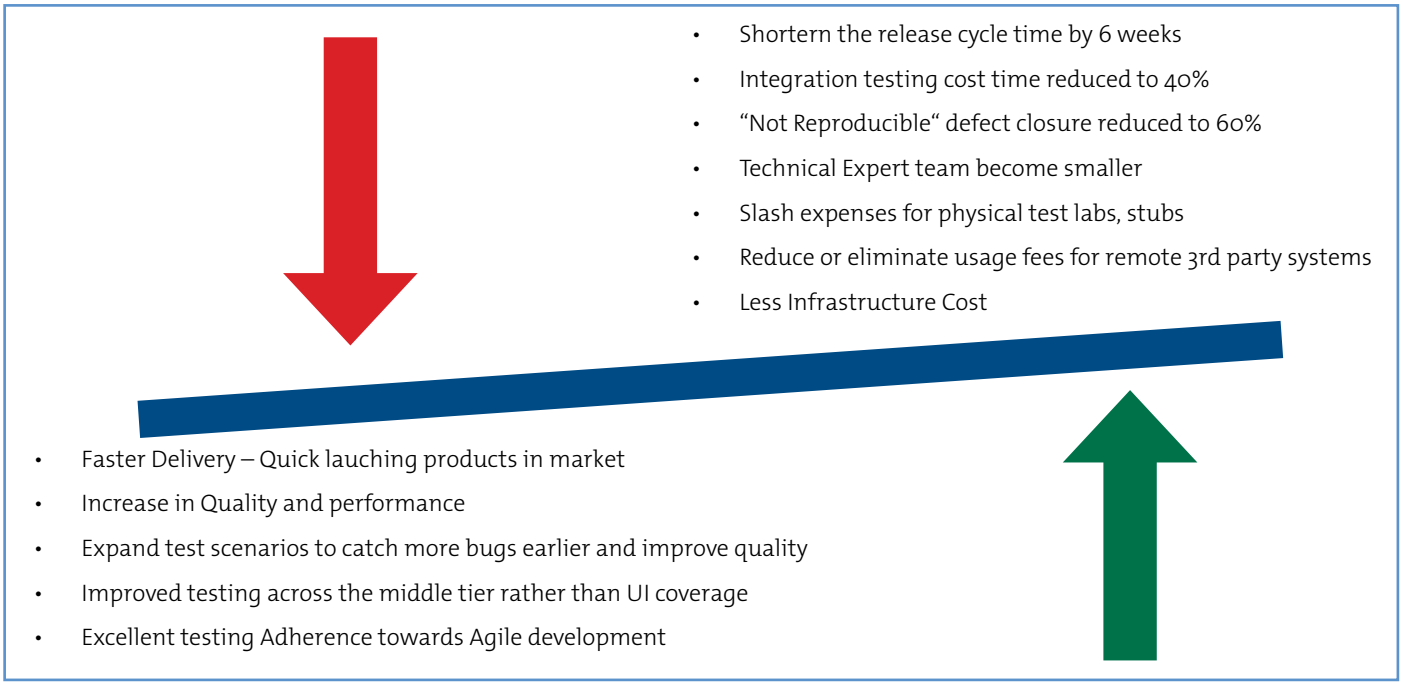


Figure 4 – Critical Success Factor

Keywords:

SOA	Service Oriented Architecture
ESB	Enterprise Service Based
VSM	Virtual Service Model
SI	Service Image
XML	eXtensible Markup Language
WSDL	Web Service Description Language
SOAP	Simple Object Access Protocol
WSDL	Web Service Description Language

Reference:

www.itko.com/lisa

http://en.wikipedia.org/wiki/Itko

http://msdn.microsoft.com/en-us/library/aa292128

> biography

Krishnachander Kaliyaperumal

works as a Test Lead at HCL Technologies and has 7 years of experience in software testing. He holds a Master Degree in Computer Applications and is a certified ISTQB professional and Six Sigma Green Belt. His prior experience in software testing includes Telecom, Aerospace, Retail and Banking domains.

Strategic investments in tough times – where testing organizations should invest

by Klaus Haller

Harry Potter is lucky. He can always look into a crystal ball when he needs more information. He sees scenes from the past, present, and future. And as we know from the movies, the predictions are correct. Predicting the future of testing is more challenging. First, I want to explain the scene I see in my crystal ball. I do not focus on methodologies such as Agile versus V-model, on tools, or testing domains. I do not focus on test consulting or test service providers. I focus on the testing organization in banks (many statements are valid for other sectors, too). Banks with a state-of-the art IT organization rely on a central test organization, often with one head of testing with few or many disciples: test managers, test engineers etc. They provide services for the IT department and the bank. In return, they get funding. In this article, I point out, first, structural changes in banks and their IT departments in the next few years. Second, I discuss the impact on the testing organization and their priorities today.

The IT infrastructure changes. There is the emerge of the cloud. There is the concept of service-oriented architectures (SOA). There is the more hidden, but high-impact trend towards multi-tenant applications. Multi-tenancy is about one application installation serving ten, hundred, thousand, or millions of customers. The application shields the customers and their data from each other. Multi-tenancy allows for economies of scale. Swisscom, for example, hosts core-banking installations for various banks. One large group of them, though legally independent, shares one installation. Such **economies of scale of multi-tenancy** change IT. They change the provisioning of IT and business services. The cloud (e.g., Amazon Elastic Compute Cloud or Windows Azure) provides unlimited scalability. If a multi-tenant application is implemented for the cloud, it scales if the customer base doubles or multiplies by thousand. Also, SOA and web services play their role. They are the glue for coupling a bank with various service providers and their multi-tenant systems.

SOA and multi-tenancy enable outsourcing applications to various vendors. This catalyzes the **commoditization of IT services**, a trend ongoing for decades. There are five stages: custom software, commercial-off-the-shelf software (COTS), application service provisioning (ASP), full application service provisioning (Full-ASP), and business service provisioning (BSP). The tasks associated with each application or process are (see Figure 1), first, the task **business requirements** with the business-IT-alignment aspect. Second, the engineering tasks are *system requirements, architecture, design & specification, and coding*. This reflects classic software development, e.g., with J2EE. It subsumes also the parameterization for COTS such as SAP. Third, the testing related tasks are *component (unit) test, component (unit) integration test, system integration test, and acceptance test*. We added *business readiness test* as a check whether the ASP or BSP provider fulfills the promised services. The task *application management* reflects managing the lifecycle and various releases, running the software on a server, and providing user support. Finally, *business process execution* is executing the business activities, e.g., bank transfers submitted to the bank by its clients.

It depends on the stage of the commoditization process which tasks are done in-house:

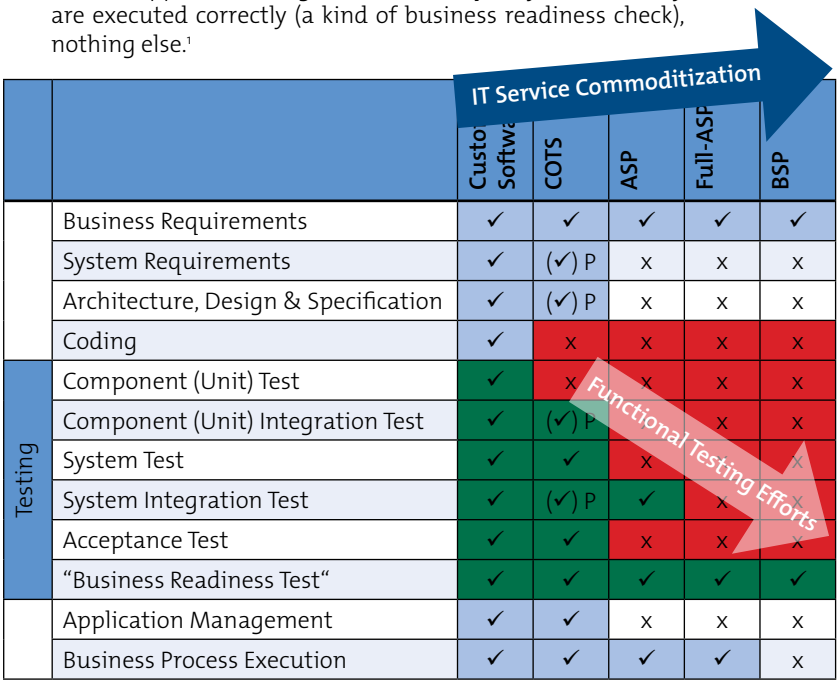
- Custom software: All tasks are done in-house.
- COTS: The bank buys standard software, e.g., a core-banking-system or a printing system from a vendor. There is no in-house software development. Only parameterization is needed and the integration into the IT landscape.
- Application Service Provisioning (ASP): The service provider implements and configures the / one application and provides the application management. The bank defines only the business requirements and integrates the ASP services into its IT landscape via software interfaces.
- Full Application Service Provisioning (Full-ASP): In contrast to ASP, the service provider manages the complete IT infrastructure and all interfaces of the bank. The bank defines only the business requirements.
- Business Service Provisioning (BSP): The bank does not see the application used for the business process. Even the execution of the business processes is now outsourced.

There is one reason banks push commoditization forward: the application or process has no strategic value. Some years ago, Carr asked and stated "Does IT matter?" and "IT doesn't matter" [1] , [2]. Today, we ask "Does this application/process matter?". In other words: Does the application or process help to impress your customers ("differentiator")? Or is the best you can achieve not to annoy him ("commodity")? A bank transfer must work. Nobody

applauds if a bank performs bank transfers correctly. But everybody complains heavily if one single bank transfer fails. So payment applications and processes are commodities. Not surprisingly, Deutsche Bank outsourced this area [3]. To give an example for a differentiator: a tax consulting application supporting bank staff when advising wealthy customers.

Commodity processes and applications are under threat for further commoditization towards ASP and BSP. When the bank starts grouping their processes into differentiators and commodities or when the IT department do this for their applications, this can be the beginning of a new round. All progress in commoditization has **consequences for the testing** (Figure 1, red and green cells):

- Custom software to COTS: Component and component integration tests become obsolete, such as the development itself. These are development tasks, i.e., this does not affect the testing organization. Developers perform such tasks, i.e., there is no influence on the testing organization. The system integration test for testing the interfaces and integration into the IT landscape remains with the testing. However, system and acceptance tests need less effort. They test only the COTS parameterization. They do not test the correctness of the complete application. Also, we observed in core-banking implementation projects that consultants of the parameterization team take over certain tests. They work highly iteratively and are responsible for requirements, parameterization, and some tests. This leaves less work for classic testing teams. It might be marginalized to setting up automatic regression tests.
- From COTS to ASP: Acceptance tests become obsolete. Clarifying the service and features is part of the selection of the ASP provider. What remains is the system integration test. It checks how the ASP system collaborates with the bank's IT landscape. There is also a need for a business readiness test, though the business can perform parts of it by itself.
- From ASP to Full-ASP: The complete IT is outsourced. No classic tests are left, not even system integration tests. There might be only some business readiness tests left.
- From ASP/Full-ASP to BSP: The bank simply routes all processes to a supplier. One might make some quality checks if they are executed correctly (a kind of business readiness check), nothing else.[1]



		Custo Softwa	COTS	ASP	Full-ASP	BSP
	Business Requirements	✓	✓	✓	✓	✓
	System Requirements	✓	(✓) P	x	x	x
	Architecture, Design & Specification	✓	(✓) P	x	x	x
	Coding	✓	x	x	x	x
Testing	Component (Unit) Test	✓	x	x	x	x
Testing	Component (Unit) Integration Test	✓	(✓) P	x	x	x
Testing	System Test	✓	✓	x	x	x
Testing	System Integration Test	✓	(✓) P	✓	x	x
Testing	Acceptance Test	✓	✓	x	x	x
Testing	"Business Readiness Test"	✓	✓	✓	✓	✓
	Application Management	✓	✓	x	x	x
	Business Process Execution	✓	✓	✓	✓	x

Figure 1: Commoditization of services and the input of testing: ✓ task to be performed internally, ✓(P) tasks performed internally, for parameterization only (not the application), ✓ no internal activities. The tasks are based (mostly) on the V-model.

Our insights into the financial industries show that many small banks are (mostly) on the "COTS" level. For larger ones, the "Custom Software" stage is still strong. The trend towards ASP or BSP is slowly gaining momentum. More ASP and BSP means a storm, i.e., a drop in work to come for (functional) testing. And now, after the discourse on IT trends and corporate strategy, I am at the heart of strategic investment decisions in the testing organization. I talk about **managing downsizing**. No testing organization wants to learn they invested into the testing of applications and processes which become obsolete due to ASP or BSP. Thus, each head of testing must know the future core-business of the bank. Next, he must identify applications and processes that are differentiators. If he has to cut costs and staff, he is not forced into hair-cut-style or random cuts. He knows where we want to invest and to improve proactively. He also knows where to invest only the minimum or to cut costs.

The future of functional testing is not a grow story. But there is hope for testing in other areas. This demands **redefining the mission of the testing organization**. Is the aim of the testing organization to deliver efficient testing services at adequate costs? Or is the soul of testing being keeper of the Grail of a functioning IT system that is not harmed by (newly deployed) software?

Load and performance testing are well established in many testing organizations. It is an example for a niche the testing organization has occupied. More niches exist. There is **data privacy**: companies do not want that testers or certain users see data they are not supposed to see. Who is taking care of this topic? The legal department or testing? **Synthetic test data** or **masking production data** are closely related [5]. Another topic is **compliance testing** (also named "regulation testing") [6]. It checks whether legal or company-internal regulations are enforced in or with the help of IT systems. Should the compliance officer, auditors, or testing take care of this topic? Compliance testing might be the next big thing due to outsourcing. Banks must check their outsourcing providers. Outsourcing providers should respect banking regulations and codes-of-conduct of the bank.

Anybody thinking that he/she can ignore what their suppliers do, should remember Apple, Dell, and HP. They get bad press due to suicides of Chinese workers at Foxconn, one of their main suppliers [7]. Banks are in a worse position. There are not only customers, they also deal with regulators.

Another niche is **security** or **penetration testing**. Today, this is often done outside the testing organization. Is the lobby of the testing organization not strong enough? Or is testing not interested? And who takes care of SLAs, such as response times in case of ASP and BSP? Is it done by the **monitoring** team in the server group, because these tests are continuously performed– or are there synergies with the load and performance testing team?

1 *A remark to BSP and Full-ASP. There is Full-ASP, meaning that a company gets rid of (nearly) its complete IT*

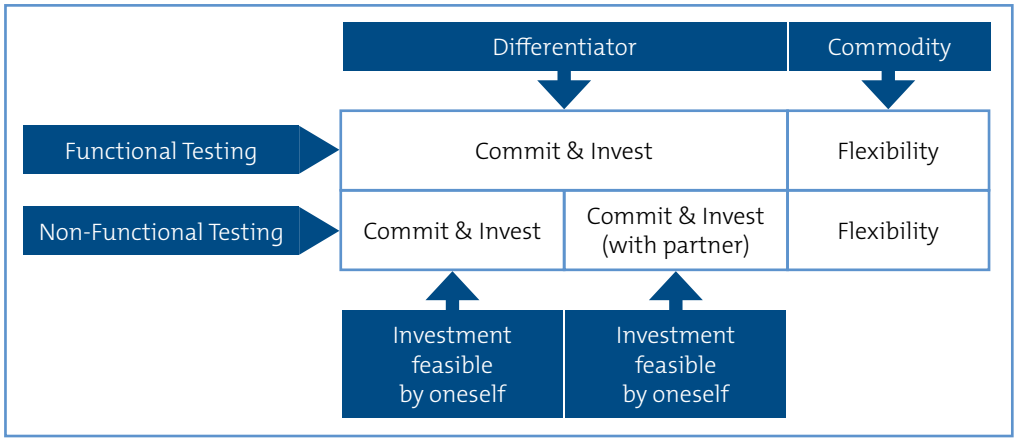


Figure 2: Strategic Investment Matrix for Testing Organizations – Commit & Invest: proactive maintenance of test artefacts, high ration of internal testers, Flexibility: low ration of internal testers and/or emphasis on outsourcing

What is exactly going to happen in the area of non-functional testing and quality assurance? This question is the point at which to remember Harry Potter and his crystal ball. The outlook for functional testing is pretty clear. The non-functional aspects are blurred. And, as in the movies, the crystal ball becomes suddenly dark. Harry has to start his next adventure. And so does the head of testing; he has to make every-day operational decisions. They shape indirectly the future of the testing organization. **Our strategic investment matrix for testing organizations** (Figure 2) might help.

However, one point must be clear: If an area requests testing services, the testing organization must provide them. They must keep the software quality high and the IT landscape stable. But even in such situations, the testing organization has two options: investing proactively or acting in a more reactive way. Commit & invest means staffing projects with internal testers. They develop and keep their domain and application know-how. It means maintaining the testing artifacts proactively. It means improving the test processes and testing tools. The other option, *flexibility*, avoids long-term investments. It allows for quick and easy cost-cuts when the work vanishes. The key is outsourcing and/or a high ration of external testers.

The testing organization has little influence on what to test, but there is always the choice to *commit & invest* or to stay flexible. In case of functional testing, the strategic investment matrix suggests:

- A process or application that is a differentiator is a candidate for *commit & invest*: proactively ensuring internal know-how by a high ration of internal testers, up-to-date test artifacts, and state-of-the art tooling.
- If it is a commodity, the commoditization might move on. This is a negative mid- to long term outlook for test efforts. The testing organization must still ensure the quality of the software[2], but there are no long-term investments. Flexibility is key. A high ratio of external testers or outsourcing testing eases shutting down test teams if the commoditization moves on to ASP and BSP.

In case of non-functional testing, the absolute costs influence decisions. In contrast to functional testing, there are nearly no synergies with the banking business. Instead, testers need more and more specialized technical know-how. Thus, certain non-functional testing areas are too costly for a bank to invest alone. We recommend:

- If the service is a commodity, it is the first candidate for the *flexibility* model, i.e. a high ratio of external staff and outsourcing.
- If the service is a differentiator and the bank can afford the investment, the testing organization should *commit & invest*. The bank can build up this topic on its own.
- If the service is a differentiator but the costs are too high for one bank, the bank should *commit & invest*. Due to the costs, the bank should look for an *innovation partner* and run the project with him.

These advices and the underlying matrix help to prevent hair-cut-style or random cuts in tough times. In good times, the matrix helps investing into projects with long-term benefits. But the matrix only helps if the testing organization does some homework. It must, first, understand the business strategy of the bank. Second, it must categorize their testing areas into differentiators and commodities.

My personal expectation for testing is **downsizing and border war**: (controlled) downsizing for functional testing and "border wars" within the IT department or with legal and compliance offices. They must clarify who is responsible for which non-functional test.

Acknowledgments: The author would like to thank Hans-Joachim Lotzer and Tjeerd Olk for the valuable discussions.

[1] Carr, N.: *Does IT Matter? Information Technology and the Corrosion of Competitive Advantage*, Harvard Business School Publishing Corporation, Boston, MA, 2004

[2] Carr, N.: *IT Doesn't Matter*, Harvard Business Review, May 2003

[3] *Deutsche und Dresdner lagern Zahlungsverkehr komplett aus*, manager magazin, 01.10.2003 (in German)

[4] Strassmann, P.: *The Xerox Tragedy*, in: Computerworld, November 6, 2000

[5] Haller, K.: *The test data challenge for database-driven applications*, 3rd Int. Workshop on Testing Database Systems (DBTest 2010), Indianapolis, IN, June 7, 2010

2 *Being able to deliver today might imply high or higher short-term investments. Reasons might be currently insufficient testing coverage or new major releases and much new functionality.*

[6] Haller, K.: Data-Privacy Assessments for Application Landscapes: A Methodology, 1st BPM Workshop on Workflow Security Audit and Certification (WfSAC), 9th Int'l Conf. on Business Process Management (BPM 11), Clermont-Ferrand, France, August 28th to September 2nd, 2011

[7] Barboza, D.: After Suicides, Scrutiny of China's Grim Factories, The New York Times, June 6th, 2010

> biography

Klaus Haller
is a Senior Consultant with Swisscom IT Services Finance in Zurich, Switzerland. He has more than six years consulting experience, mostly in the financial industries. His main interests in testing are information systems and database applications throughout their lifecycle, test data management for application landscapes, and process (re) engineering. Furthermore, he works on future testing and quality assurance services and publishes frequently in magazines and at conferences and workshops. Klaus studied at the computer science department of the Technical University of Kaiserslautern, Germany. He received his Ph.D. in databases from ETH Zürich, Switzerland. He welcomes your feedback to his article at klaus.haller@swisscom.com

The future of testing through the influence of "open source"

by Olivier Renault

In this article, I want to present a vision of the future of software testing through the development of tools and methods influenced by changes such as „open source".

The current practices of testing

How can we imagine the future of software testing and especially the future of the tester in this environment? A Google search on „the future of software testing" gives more than 175,000 responses. This indicates a lot of intense speculation on the subject, but an answer emerges through the challenges faced by professionals. Technically, the beginnings of a solution defining the new testing ecosystem seem to be emerging through the current technologies and methodologies, which are the basic elements of a software testing platform which only remains to describe the interconnections. This brings us into this new era. Did we have any awareness of that?

For years, software testing was structured around a business, which adapts to changing business development. The state-of-the-art until the turn of the century was that development projects followed the methodology of the V-cycle (or waterfall) and the testing phases were dependent on the input documents for building accurate test plans and test cases.

Unfortunately, development times lengthen while the specific dates of availability to users do not move. The qualification phase at end of the cycle often finds itself shortened and sometimes completely canceled. Software testing will suffer from longer periods of system/software development. To counter this trend, new development methodologies emerged which are known as „Agile" [1]. Projects will no longer take place in a long life cycle that is subject to these risks, but in short cycles (called sprints) that allow - with the shortening of these iterations - to deliver functionalities in a faster development and qualification process. To achieve this goal, it is necessary to involve the software testing teams early in the sprint. The purpose is to anticipate as soon as possible the testing needs, have a better control of the availability and configuration environments and to specify the tests and their implementation within a short timeframe.

This methodological change has had a fundamental impact on the profession of software testing. To manage it, some methodologies and best practices have been implemented. Like CMMI that evaluates software development practices, testing experts designed TMMi [2] that evaluates best practices of software testing. Other methodological frameworks have been created such as TMAP/TPI Next (Sogeti) [3], or certifications such as ISTQB [4] which has the merit of proposing a unified language for all of the testing community.

The implementation of Agile methods has raised many questions among testers on how to carry out their tasks. Did you know that one of the discussions with the most contributions to the thread „Quality Engineering" from sqaforums website [5] is about the „correct" volume of documentation for an Agile project? Testers who are used to a profusion of specifications and documentation on which to base their thinking tests find themselves in an Agile environment and have to completely rethink their way of working.

Testers cannot keep up other than by trying to automate as many tests as they can. Add to that the fact that testers are generally seen as the bearer of bad news which can cause some tension between developers and testers (see § 1.5 ISTQB, on the psychology of testing). With these conditions in mind, how can we foresee a better future for software testing?

Towards a new approach to testing

This will certainly follow once again the example of the changing world of development. The major change now is „crowdsourcing", which results from the maturity of open source tools and the extensive development of the Internet. These tools have become indispensable in the world of development, and it is now necessary to adapt the testing world to these new offerings. Indeed, through the values transmitted by the Internet, we are witnessing the rise of a new generation of people wishing to work in the community.

The success of open source has achieved a high degree of technicality on many products and has spread to areas other than development (e.g., knowledge, with the creation of the encyclopedia „Wikipedia"). This trend is reaching the world of testing, as shown by „weekend testing" [6], which involves a meeting of professional testers who test an open source product for a few hours in order to take advantage of their community expertise

and improve it by exchanging their know-how. While developers share their source code (social coding) with online project hosting websites, such as github or google code, such an approach has not yet developed in the world of testing.

The market has already tried to adjust to the arrival of crowdsourcing, for example with offers like „Testing as a Service“ (again, following the development world which provides „Software as a Service „) or by offering a paying access to a testing community (utest [7]).

However, even if these offers are very interesting, I see them as a „marketing package“, offering „outsourcing“ instead of „crowdsourcing“. Personally, I see the future of software testing in the development of real „crowdsourcing“ based on the implementation of Agile concepts through a set of „open source“ tools. I envisage a platform for testing industrialization oriented Continuous Integration / Test Management / Environment Management / Defects Management / Implementation and Automation of different types of testing (unit, regression, functional, performance, security, localization, usability,...). And all of this using natural language („keywords“), fully integrated into a single platform through script libraries written in a easy to handle language.

We are not talking about sharing scripts made to test specific applications (where the rights belong to companies that employ the testers) with commercial testing tools (such as HP ® or Microfocus ®), but to share test cases using „standard keywords“, or scripts to implement new keywords that can also be applicable in other contexts.

The sharing of test plans will improve the quality and relevance of the tests and provide a first set of a test repository that you can improve with no limits. For example, you might perform tests requiring specific expertise such as security testing of web applications, or load testing without mastering the skill.

The future of the art of testing could then be seen as not simply hunting bugs, but also as being the orchestra conductor in the supply of a solution which meets the needs of product testing, with all the actors of a development project. Your business users can design the test cases in natural language early in the project, these can be applied directly by the platform, and developers can practice varied and smart testing. This is in fact the practical application of recent advances in the world of development and testing.

A proposal for technical implementation

The solution I am proposing today is based on the following principle: a platform which drives testing tools (open source, commercial, scripts, etc.) with a framework oriented Behavior Driven Development (RobotFramework) from test cases written in natural language (keywords). These test cases are stored in a test management tool (TestLink), and running is triggered by a continuous integration server (Jenkins). I've named this platform 'JinFeng', a Chinese name which means 'Golden Phoenix'.

This principle can be applied with other programs (Cucumber, Salome-TMF, Fitnesse, CruiseControl, etc.) but, as shown in Figure 1, the interface between Jenkins, TestLink and RobotFramework, is based on a single engine, which is the Plugin Jenkins / TestLink [8] created by Bruno P. Kinoshita [9]. This has the immense advantage of being able to easily create and set up this platform. The other interface between the acceptance testing tool and the automation tools is done in Python. This language is particularly easy to learn for testers; in fact it is near to becoming a standard which allows us to manage any desired testing tools (Selenium for testing web application, Jmeter or Grinder for load test, Sikuli for GUI testing, etc.). And it offers native capabilities to automate many actions.

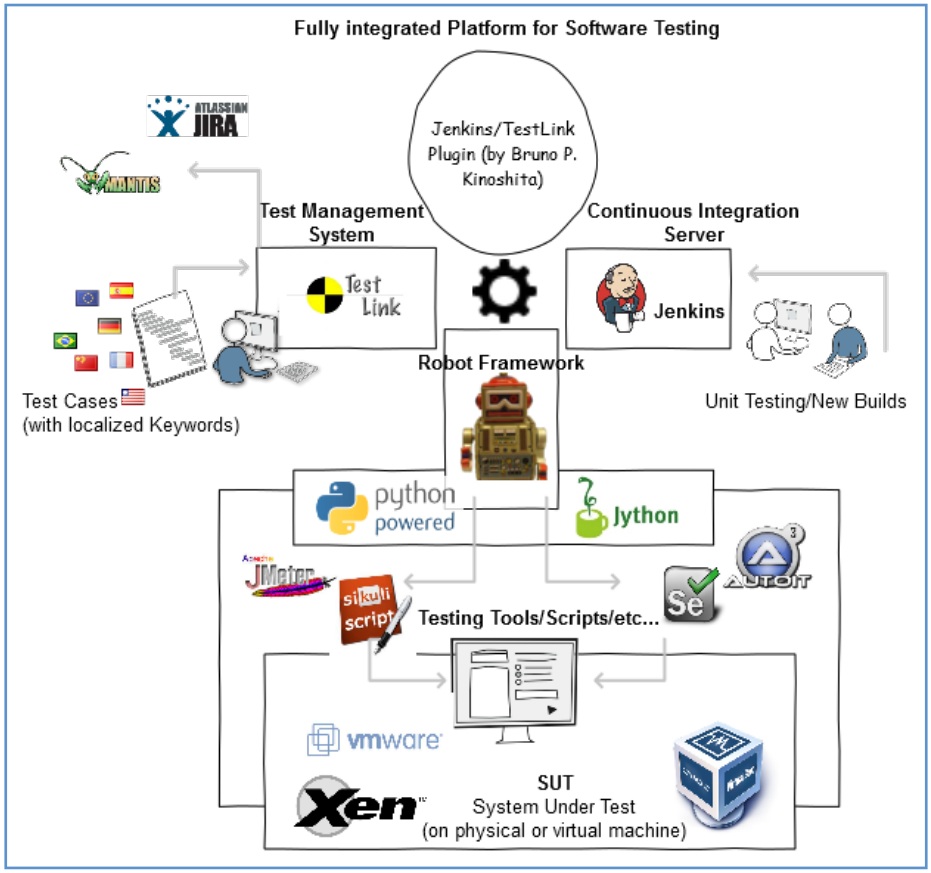


Figure 1

By adding Jython to our Python installation, we obtain the ability to drive through our keywords, any product with opened JAVA API (this is the case for example of the Vijava API for manipulating Virtual Machines from a VMware ® vcenter).

The benefits are therefore multiple, as keywords that are in the test cases can help to:

- manage test environments (create or modify VMs ...)
- automatically deploy the latest build of the AUT / SUT
- run existing scripts implemented with commercial tools
- connect to machines in SSH or Telnet
- send messages through different channels (smtp, Skype ...)
- generate output files in TAP format, which will be automatically included in TestLink (through the plugin Jenkins / TestLink).
- and many other possibilities (limited only by imagination ...)

The operating principle is to set a task in Jenkins with the use of the plugin Jenkins / TesLink. During the build of the task, the plugin will extract from TestLink all the test cases that will be designated in TestLink as „automated" and match the settings of the plugin.

For each Test Case, the plugin will then perform the actions defined, in fact calling RobotFramework by providing the test case in process (Figure 3).

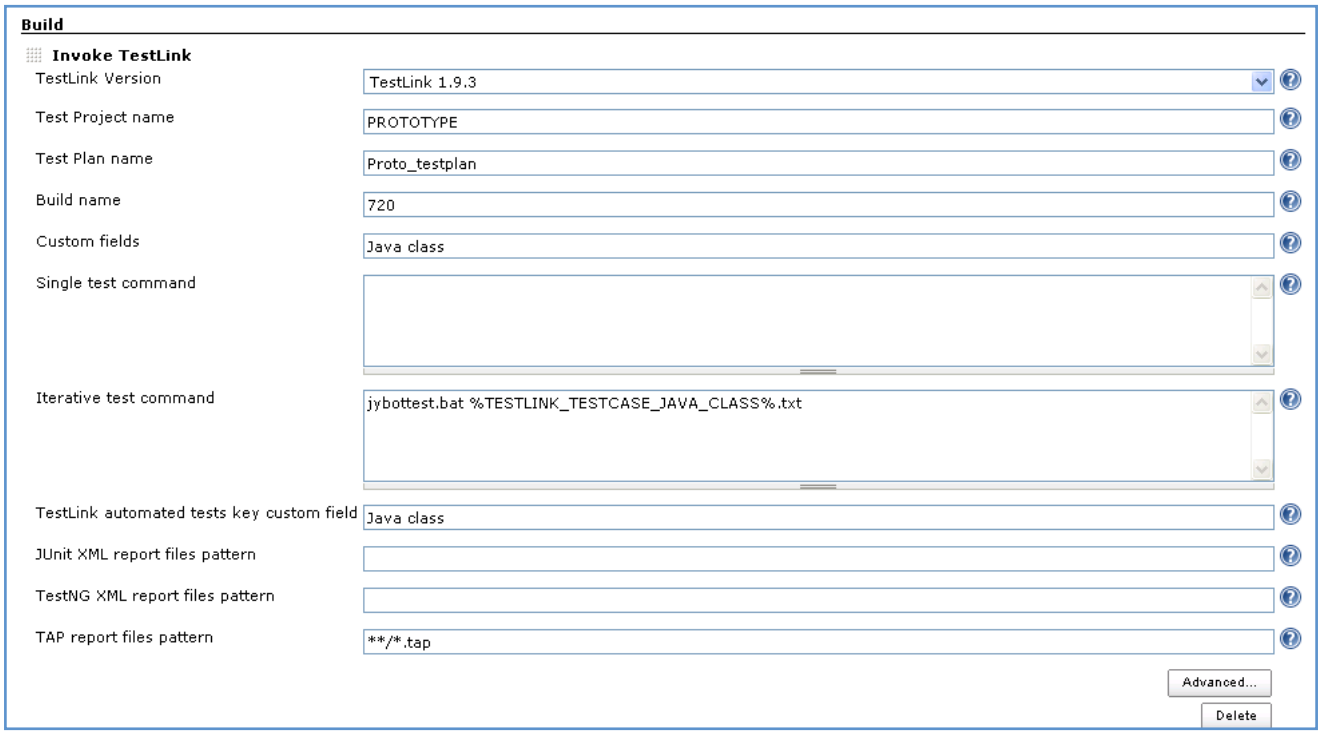


Figure 2

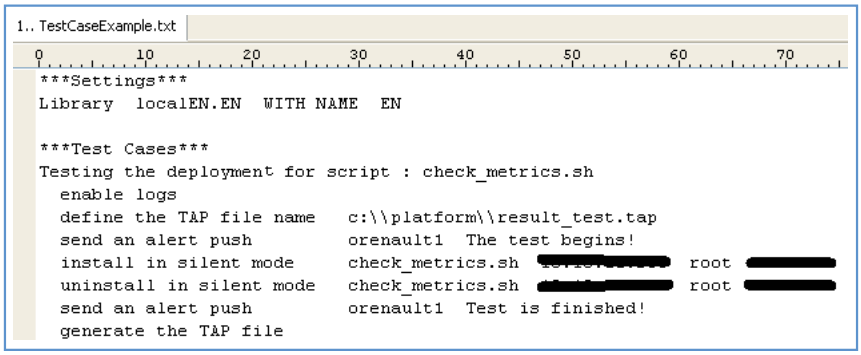


Figure 3

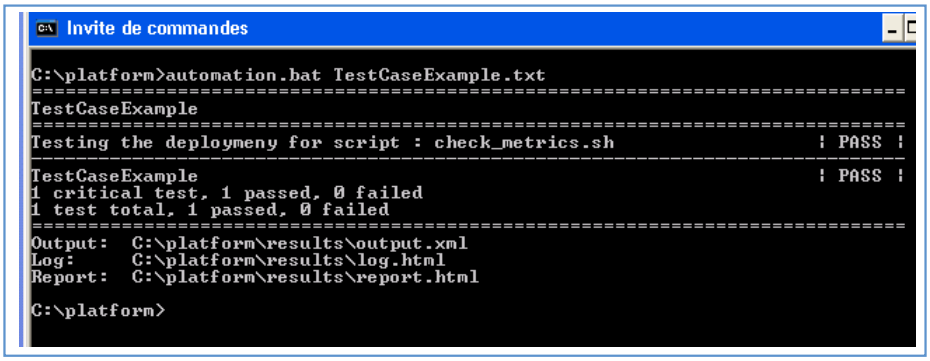


Figure 4

Sie suchen
das Besondere?

Díaz Hilterscheid

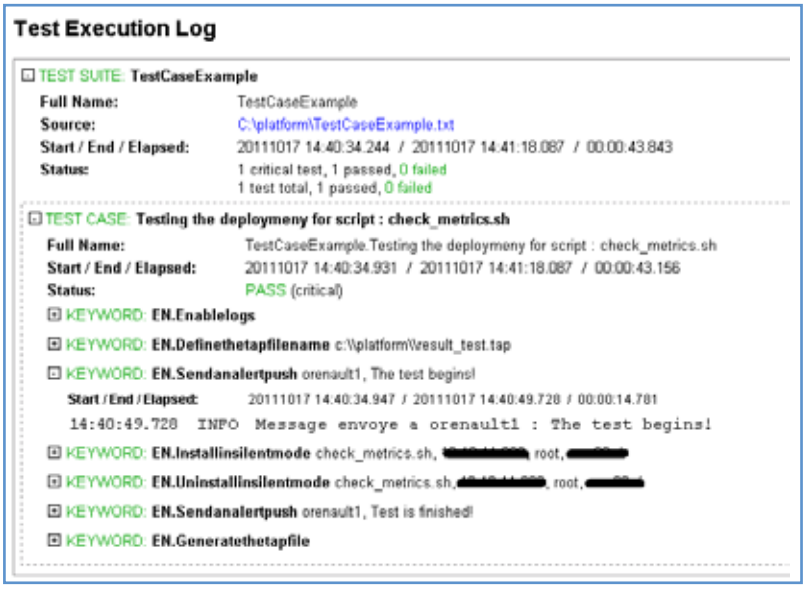


Figure 5

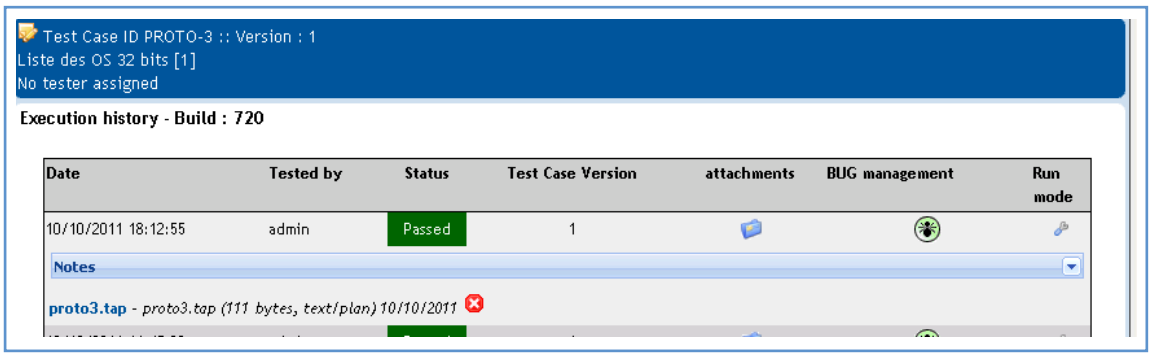


Figure 6

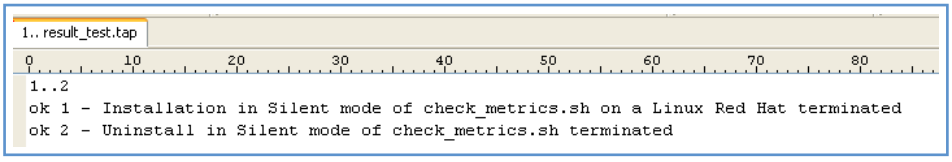


Figure 7

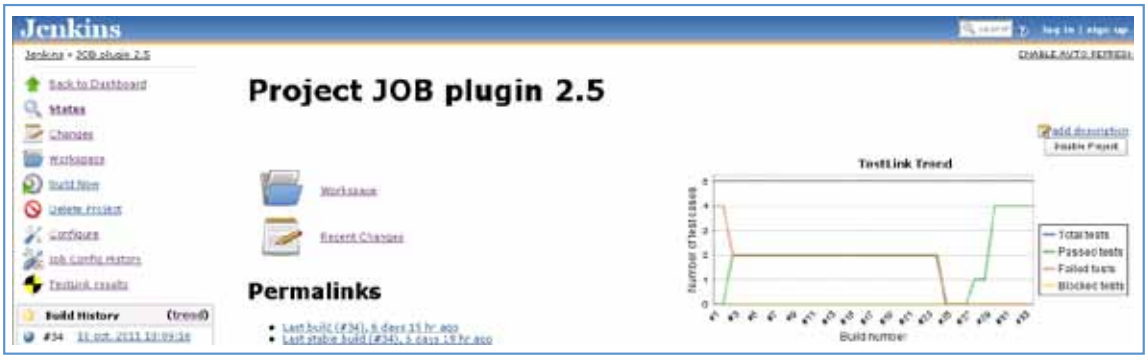


Figure 8

RobotFramework will then parse the test case and perform the actions corresponding to each keyword or sentence in natural language (Figure 4) with the data set provided in the test case. The scripting for each keyword is in the form of a Python / Jython script, contained in a standard library (core).

At the end of execution, RobotFramework will generate a report on the progress of each test case (Figure 5), while the plugin will automatically update the „Running tests" in TestLink (Figure6) to update the state of each test case as Passed / Failed. This is based on the analysis of test result files with a TAP structure (Test Anything Protocol) [10], which are created automatically via the Jython core library and tap4j [11] (Figure 7). Jenkins will also be updated on the progress of test cases (Figure 8).

The Core has been designed in a modular way to be easily extensible. The first layer contains only the keywords and the log messages in a specific language (Figure 9). It is very easy to change these keywords and messages to a different language simply by duplicating and modifying this file.

The second layer includes two Python files that describe the implementation work. The file keyword_CORE.py contains standard keywords (Figure 10), the file keyword_AUT.py includes specific keywords related to an application to be tested. And finally, the third layer includes a set of files that make up the technical implementation in the form of Python libraries. There are two more files: CORE and AUT, and also all other implementations such as skypelib, smtplib, taplib, vmwarelib, etc.

Wir auch!

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

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

Sie haben

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

Sie verfügen über

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

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

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

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

Mehr Stellenangebote finden Sie auf unserer Webseite:

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

Díaz Hilterscheid

Figure 9

Figure 10

Conclusion

The ease of installation of everything on this platform (less than a day for all components), and the ability to interface Python / jython with virtually all other products, offers really interesting and „infinite“ use and outlook.

As part of „crowdsourcing“, it would be possible to quickly develop a large number of test cases that are written in natural language and reusable in many projects; and to develop the implementation of related keywords, by changing CORE files and sharing them. For example, it is conceivable to use the open source tools Freemind or dia to drive our tests using graphs and to obtain the opposite with charts constructed by keywords. I invite you to visit http://www.sqaopen.net [12] for more examples and a tutorial detailing the implementation of the JinFeng platform.

References

[1] http://en.wikipedia.org/wiki/Agile_software_development

[2] http://www.tmmifoundation.org/html/tmmiorg.html

[3] http://www.tmap.net/en/tmap-next

[4] http://istqb.org/display/ISTQB/Home

[5] http://www.sqaforums.com

[6] http://weekendtesting.com/

[7] www.utest.com

[8] https://wiki.jenkins-ci.org/display/JENKINS/TestLink+Plugin

[9] http://www.kinoshita.eti.br/

[10] http://testanything.org/wiki/index.php/Main_Page

[11] http://www.tap4j.org/

[12] http://www.sqaopen.net

> biography

Oliver Renault

I am 39 years old and live in Paris. I have used computers since the age of 11, and I studied at Polytech'Nantes, a school of computer engineering. I have over 12 years of experience in software quality assurance, both as a consultant and manager. I worked in areas such as software publishing, banking, telecommunications, military, etc... In particular, I worked four years for Compuware on testing tools and am very familiar with the tools offered by different publishers. I think that open source software products are still not sufficiently evolved to be as efficient as commercial products. This is true for each of these tools individually, but I recently discovered that the integration of some of these tools allows to build a very powerful testing platform, and perhaps more powerful than a commercial product. So I recently created a blog to present and share this work and the little tips on integrating systems and applications that I discovered during my daily work.

Break free!

freedom learning
beyond the classroom

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

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

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

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

te testing experience

www.learntesting.com

learntesting

Relevance of Test Data Management (TDM) within the Testing Organization

by Sridhara SN & Karthik Authoor Venkata

The Philosophy of Test Data Management

Good quality test data is the key to the success of new and existing applications. The ability to create test data quickly for a range of test cases will save organizations time and money. Test Data Management (TDM) is a solution that helps to deliver correct and accurate test data for development and testing projects.

Abstract / Business case

The test data management framework can be well integrated with the QA activities performed across comprehensive testing life cycles, thus yielding optimization and efficiency of existing tools, resources and processes.

This article attempts to focus on:

1. How to map Test data workflow mapping with comprehensive testing life cycles?
2. How to map Skill set and cross utilize QA & TDM groups and resources?
3. What are the benefits of starting TDM activities at the early stages of testing life cycle validation?
4. How structured TDM helps reduce Quality Assurance effort and costs?
5. Leveraging investment and tools in Quality Assurance for TDM Quality Assurance Maturity and its impact on TDM implementation

This article is based on the actual experiences of the authors.

Introduction / background

QA and TDM – Concept introduction, intersection and divergence

Test Data Management deals with ensuring the availability of the right amount of the right kind of data at the right time in the non-production environments. The data so provided should also not compromise on security and privacy.

Today's enterprise QA and testing processes involve or use structured testing procedures that account for around 50 to 60% of the application life cycle. Using a comprehensive Test Data Management strategy, QA organizations can provide a consistent and standardized approach to ensuring high quality and reliability of large enterprise applications and systems.

Test data management does not deal with specific test cases, but rather with the test data (creation / maintenance / management / enhancement/ provisioning) going into specific test cases for specific test scenarios. In order to acquire good quality test data, and manage it effectively, test data management should be an important part of the company's overall QA/testing strategy.

Robust test data management processes are essential in maintaining applications and databases. In addition, the recent rise in identity theft, industry regulations and legal compliance continues to put pressure on organizations using very rudimentary techniques to generate, provide and manage test data.

Problem statement

TDM not aligned with QA in an organization

Speed of business and growing competition requires IT organizations to deliver mission-critical enterprise applications in-time, with high predictability. To achieve this, QA organizations need to create a reliable testing process that improves application quality, reduces time-to-market and minimizes the cost of development and testing. Test Data Management caters to the various test data needs for development / enhancement / maintenance and testing of applications.

Costly application development time is lost to the cumbersome process of test data management. Pulling data from "live" databases is unsafe, requires additional time to run the test due to file sizes and uses up costly and/or scarce CPU. While TDM will directly benefit an organization's QA process, integration with its existing QA practices (across comprehensive testing life cycle stages) is equally important for optimization.

Introduction of solution

Defining a TDM framework - Mapping test data workflow with comprehensive testing life cycles

Testing Life Cycle	TDM Phases	Entry Criteria	Activities	Exit Criteria	Measurements	Deliverables
Requirements testability review	Test data requirements gathering	1. Understand customer high level expectations 2. Background reading of customer's business profile 3. Understand customer's domain of work 4. Thorough understanding of testing process, methodology and applicable standard/guidelines 5. Familiarity with test data management tools	1. Understand the test data requirements 2. Get an idea of the scope of testing 3. Understand data model, data Flow, data sources and different file types used for storing data. 4. Understand data storage – metadata, transaction data, etc. 5. Understand data archiving and retrieval mechanism 6. Understand data Refresh and clean up 7. Identify test data management focus areas 8. Identify Single Points of Contact (SPOCs) 9. Identify and study environments 10. Understand technology and application landscape 11. Analyze test data risks and issues 12. Understand release management	Understanding test data requirements (software / hardware)	1. Test data requirements gathering effort and schedule 2. Review defects in the test data requirements document 3. Number of re-work iterations of the test data requirements document	1. Test data requirement document 2. Test data request form 3. Test data landscape document 4. Governance structure
Test strategy	Set test data management strategy	1. Understand the scope of testing 2. Understand customer business requirements and expectations 3. Test data management requirements gathering completed	1. Assimilate the scope of testing and prioritize test data provisioning 2. Prepare Operation Level Agreement (OLA) and Service Level Agreement (SLA) documents 3. Prepare & submit test data management strategy document. This includes: • Identify different methods of making test Data available • Data sub-setting • Data creation/extraction • Data reuse • Data provisioning • Data reservation • Data sampling • Data masking • Data refresh • Data archiving and retrieval • Tool analysis and finalization. 4. Creation of test data distribution log 5. Identify estimation methodology, skill sets required. 6. Identify metrics. 7. Sign off from client for submitted documents 8. Scope and create a detailed project plan	1. Review and sign-off test data management strategy document 2. Review and sign-off SLA document 3. Detailed project plan	Effort Schedule Review defects	1. Test data management strategy document 2. SLA document 3. Detailed project plan 4. Test data distribution log

Testing Life Cycle	TDM Phases	Entry Criteria	Activities	Exit Criteria	Measurements	Deliverables
Test planning & design	Test data Planning	1. Test data requirements specification 2. Test data management strategy	1. Understand data requirements needed for testing 2. Define workflow for test data requisition 3. Prepare test data plan and obtain sign off 4. Create test data model (DFDs, entity charts, business and stakeholder relationships) 5. Identify gaps between requirements and current test beds and create test data profile document 6. Study existing data; enter them in test data catalog with version number. 7. Prepare test data traceability matrix.	1. Test data plan and test data traceability matrix prepared, reviewed and reworked 2. Review defects are tracked to closure	1. Test data plan, preparation effort and schedule 2. Review effort 3. Test data plan rework effort 4. Defects injected and detected	1. Test data plan 2. Test data traceability matrix 3. Test data model 4. Test Data profile document 5. Test data catalog 6. Test data requisition workflow
Test Execution	Test data creation / extraction	1. Test data plan, test cases are available 2. Test environment is ready 3. System architecture and design documents	1. Implement finalized tool as applicable 2. Perform test data creation / extraction activities • Analyze if data can be provided from existing test bed or needs to be created • Wherever applicable, determine quantity of production data to move to test environment and create one or many smaller copies of the database such that the smaller sub-set should have required data to validate end-to-end test cases • Mask the sub-set data as required, and make a Gold Copy of the masked data • Validate if data confirms to regulatory compliance • Load data dump to target region • Communicate data readiness to stakeholders 3. Perform test data validation 4. Identify data that can be reused or shared 5. Take backup of new data once set up 6. Assign version number to the backup for easy retrieval 7. Update test data distribution log 8. Track progress regarding OLAs and SLAs 9. Update test data traceability matrix 10. Perform data quality analysis 11. Capture, analyze metrics 12. Conduct triage on test data bugs 13. Verify bugs on fix and track to closure	All test data provisioned as per test data plan. Data masking implemented as applicable	1. Test data creation/ extraction effort 2. Schedule 3. Defects	1. Physical test data / gold copy database 2. Test data distribution log
Test Evaluation	Test data reporting	1. Test cycle completion report generated 2. Client accepts the work products	1. Do variance analysis from test data plans 2. Do test data coverage variance 3. Submit metrics analysis report	1. Metrics analysis report is submitted 2. Project learning and best practices documented	Effort	Project learning and best practices document

Application of solution

Benefits of starting TDM activities at the early stages in the testing life cycle validation

- Testing gaps from a test process and test data perspective should be plugged at an early stage. This would help save huge testing effort and test data management effort spent if detected at a later part of testing life cycle.
- Enables early detection of defects in the testing life cycle
- Optimizes testing process and cross utilization of QA & TDM resources
- Optimization of testing at an early stage will not only help easy movement to testing, but also TDM process maturity path
- Will streamline testing and test data management life cycle workflow
- While the testing team is doing requirement testability analysis, the TDM team can validate the test data requirements. This will help in planning for the right kind of data upfront and will also help to identify the requirements for generating synthetic test data for the various environments, including the rules, size of data etc.
- During the design stage, when the testing team creates test models (using the model-based testing approach), the TDM team can create test data models (using DFDs, entity charts, business and stakeholder relationships). This helps in understanding the referential integrity needs across various data bases much earlier in the life cycle.

Structured TDM helps reduce Quality Assurance effort and costs

A structured test data management process helps to manage enterprise application data in the testing environment and delivers dramatic results including improved reliability, faster time-to-market, reduced development costs and higher quality.

In short, a structured enterprise test data management helps you do more with less. For the following test data management focus areas, we have provided various options to choose what is best suited for your scenario, rather than a single generalized approach.

- Test data subsetting
- Test data reuse

Test data subsetting:

Large volumes of test data, whilst necessary for stress or load testing, inhibit functional or regression testing. Wherever possible, a sub-set is more effective and is easier to validate. However, it is not a simple case of taking a certain percentage of records, since referential integrity as well as scenario needs have to be maintained.

In the following table, we have listed the various options that are available when test data subsetting is considered. However, when subsetting is not a need, we recommend synthetic test data generation as the best option.

Options	Advantages	Disadvantages	Suited best under these specific testing situations
Taking a full copy of the production database	High probability of having data for all types of test scenarios since production data is exhaustive.	• May not cater to specific negative scenarios which have been created using specific test design techniques. • Too large, taking up space and extending processing time • Time consuming to create the test environment • May contain sensitive or personal data	When testing needs to be carried out in an exhaustive manner
A calculated subset of production data taking into consideration various factors like the number of applications in the release and testing priorities as stated in the test strategy for testing these applications, the interactions between these applications and other external entities, etc.	• A good amount of required data might exist. • Easier to validate since we have a smaller database. • Lowers infrastructure costs like DB cost, hardware cost, labor costs etc.	• Difficult to build a subset which maintains referential integrity • Needs rules and good amount of thought as to what to extract/include • May contain sensitive or personal data which might be against the prevailing laws of data security	When testing is specific to a release / specific enhancement
Blank / empty files or database populated with required data	• Required volume of data can be populated with thorough knowledge of the application data needs. • Data refresh / cleanup efforts can be minimal	• Time consuming and labor intensive • Needs rules and a good amount of thought as to what to include • Does not contain real data found in production	When specific testing scenarios need to be tested or rare test situations need to be simulated

Options	Advantages	Disadvantages	Suited best under these specific testing situations
Synthetic creation of all types of test data (like meta data, operational data, configuration data, etc.) – Very structured and matured process in the field of test data management	• Automation can be explored to enhance the test data generation efficiency thereby aiding faster test case execution. • Data privacy issues can be overcome since the data used would always be fictitious. Hence lots of time can be saved due to data security policy monitoring and implementations. • Can cater to positive and negative testing, since data can be created to suit any type of test requirements	• Difficult to create with referential integrity • Does not contain real data found in production	When all types of testing like functional, regression, patch releases are in the scope

Test data reuse:

Reusability across cycles: Reusing test data across cycles is an effective strategy to optimize test data setup effort. Reuse opportunities are identified during the test planning phase (test data management "strategy" phase) by the test team. As part of cycle planning, when test cases/conditions are nominated for execution in multiple cycles, the test team should analyze if the associated data requirements will remain usable after the test case has been executed (passed/failed). Test teams will thus optimize test data requirements at the time of planning.

Reusability across test phases: Test data can be reused across different phases of testing if these phases are executed in the same environment. The issue in adopting this approach is that it risks the exclusivity of test data between two independent testing teams. The test data team should verify all aspects and ensure that the common usage of test data does not impact execution of either team. Given these difficulties, the strategy should be used sparingly and in situations where the overall data requirements volume is very high and the available setup time is not sufficient.

Future direction / Long-term focus

Skillset mapping and cross utilization of QA & TDM groups and resources

Skillset mapping and cross utilization of QA & TDM groups to leverage on existing investment and better utilization of available FTEs

Role/Focus areas		QA Team	TDM Team
Database Design	Schema design	B	I / E
	Data modeling	B	I / E
Data Storage	Capacity planning	NA	I
Data Security	DB security concepts	B	I / E
Data Analysis	Data profiling	I	I / E
	Data quality analysis	I	I / E
Data Stores		I	I / E
DBMS Concepts		I	I / E
Test Data Management (TDM)	TDM processes	B	E
	TDM tools usage	B	E
	TDM tools installation / configuration	B	E
	Tool customization	B	E
	Data masking tools	B	B
SQL queries / procedure review (including tools)		I / E	E
Test management tools		E	I
Domain exposure		E	I
Regulatory norms		B	E
Testing processes & methodology		E	I

LEGEND:

E – Expert: Expertise in the specific area, hands-on experience, demonstrates high maturity in terms of end-to-end processes, can influence decision making

I - Intermediate: Good knowledge, hands-on experience

B – Basic: Aware of basic concepts, trained without hands-on experience

Results / Conclusion

Leveraging investment and tools in Quality Assurance for TDM
Quality Assurance maturity and its impact on TDM implementation

Usage of Quality Center for test data management:

- The approach utilizes the existing resources (QC is already being used for test management). It provides a multi-user access solution and facilitates centralized test data management, tracking and reporting for all testing streams in a release.
- The approach has the potential of significantly saving the manual effort that would otherwise be required in a spreadsheet based TDM approach. In the latter approach, data collection from multiple sources, synchronization, distribution, etc. can be a challenge.
- It provides a real-time tracking of the data request status and any changes made to it.
- It facilitates effective daily status reporting including generation of custom reports for various stakeholders.
- Security and customization can be enabled using the admin functionality of QC, i.e. the type of fields, security access to different stakeholders, etc.
- Batch files can be attached to data requirements as necessary.
- Auto-notification mails can be configured on completion / blockages.
- Testing work flow tools can be integrated between testing groups/ life cycle and TDM groups / process.
- The test data repository used for Testing and Test Data Management can be a common database integrated with the common Testing/TDM workflow tool. This will prevent investment costs in disk space and workflow tool costs.
- Traceability between test data and test cases will be established creating a custom attribute ("Data definition - DDEF #") in the quality center. This field will be created in test plans and will be a test case property. The DDEF # field will also be a primary field in the test data request form. The value of DDEF# for a test case in the quality center will identify the corresponding test data request in the data management project in the QC.
- Test data for reuse can be identified by using data definition (DDEF #), which could be captured as an attribute at the test data request level. The QC data request form will contain a field to capture the reusability flag that will indicate if the data for a particular request can be reused in the next cycle. This flag can be modified again in the execution phase if the data gets corrupted.
- The TDM process needs to get aligned with the defect management process to analyze whether a defect requires a code fix, a data fix or an environment fix. This is deemed as a much matured process and depicts the handshake between QA team and Test Data Management team.

TDM implementation best practices:

Test Data Management Focus Areas	Best Practices
Test data generation	• Before modeling the test data, perform a thorough analysis of business requirements. This helps identify loopholes at an early stage and reduces rework. • Reuse of data should be explored before actual generation. • Implement defect prevention methodology to ensure good quality data delivery.
Test data management process	• Use of templates should be encouraged so as to track the data requests in an efficient manner. • Prepare generic test data templates in the TDM tool ensuring maximum reusability. This enables serving similar requests for multiple test cycles. • Versioning of test data templates helps cater to multiple releases of test data for the entire testing cycle. • Define a standardized process flow for the complete test data development life cycle. • Need to have a standard process for data reservation. • Development and test environments to be different for different applications. • A comprehensive data dictionary across all business applications should be created and maintained for establishing relationships between business applications and mainframe /distributed environments.
Test data reuse	• Define and implement test data backup procedures. • Should have an archiving process in place. • Process should be defined for database lining.

> biography

Sridhara SN
comes with a wide variety of delivery experience in handling multiple development and testing projects. He has close to 14 years of experience in the software industry. At Infosys Limited, he is a Senior Manager and works for the Financial Services & Insurance group with a focus on specialized services delivery.

Karthik Authoor Venkata
comes with extensive hands-on experience in testing. He has 13+ years of industry experience in software testing, release and configuration management. At Infosys Limited he is a Senior Manager and works for the Financial Services & Insurance group with a focus on specialized services and test data management. His special interests include test design techniques.

Testing in the 22nd century

by Thomas Hijl

"No one would have believed that in the last years of the 21st century, human affairs were being watched from the timeless worlds of space."

Many readers my age (and older) will recognize this sentence from the year 1978. The unexpected happened. Our world was attacked by another world. I wouldn't want to make a comparison here with our world of testing, but still "We are under attack." by developers and designers. Not in the sense that they find us too critical verifying their deliverables. No, they are improving their work and their way of working and that's a threat to us. As long as they make mistakes, we have a job. The more they improve, the more our work will become obsolete.

In 2100 there might be a time where designs will be verified completely automated before anything gets build. The developers might have sophisticated tools to verify directly what they have built. To see whether it suits the infallible design and correct their mistakes immediately. Think of the consequences for us testers!

I predict that in the "near" future, there won't be many functional testers anymore. We'll be left with automated testing and the remaining part of user acceptance testing.

In the less near future we'll have automated designing with virtual reality, and building will be mostly prefabricated modules molded together with customer specific exterior.

Am I just guessing? That's up to you to judge. However, let's zoom in on the test aspects of the year 2100, when some of the predictions made today will come true.

Test base:

There won't be any test base.

Test object:

There won't be any test object.

Test ware:

There won't be any test ware anymore.

Mmm, that doesn't look good for us testers. But we're not the only ones.

The fact that there won't be any testing anymore might not be the result of designers' and developers' improvements, but because of a perfect storm, a nuclear or chemical war or an extra-terrestrial invasion. *"No one would have believed ...."* might well be true.

"The chances of anything coming from Mars are a million to one he says, ... but still they come."

> biography

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

Experiences in Automating Requirements Based Testing

by Dr. Mike Bartley

When buying a new car it is usually simple to state your requirements: "*I want it to go from 0 – 60 mph*[1] *in 8 seconds*"; "*I want it to give 50 mpg at a steady speed of 60 mph*" and "*I want it to be yellow*". It is also easy to check your car meets these requirements. For example, there are numerous websites on acceleration and fuel efficiency for every type of car. You can even get technical on the color. However, such simple consumer requirements become harder when we turn to technology. I recently needed to buy a laptop that could deliver both PowerPoint and Linux demonstrations to two separate screens. The websites tried to sell me on CPU and memory which I found hard to translate back into my requirements.

When it comes to software, requirements become even more difficult to state and test products against. In support of all of our experiences, there are a number of surveys[2] that attest to the fact that many projects fail (missed deadlines; budgets exceeded; requirements not met) due to the difficulty in accurately record and maintain user requirements. Indeed, it is well known that users have trouble to define their requirements.

My company, TVS, works with a number of companies to help them test their software and verify their hardware. They capture their requirements in a variety of formats: documents, use cases, user stories, specification by examples, specifications as tests. Some of them use requirements tracing which gives them the ability to follow the life of a requirement, in both a backward and forward direction.

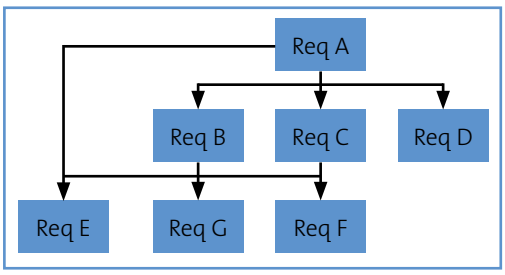


Figure 1: Bi-directional requirements tree

1 *For those fully metric, "mph" means "miles per hour" and "mpg" is "miles per gallon". You can replace these with "metres per second" and "kilometres per litre" without any loss of understanding.*

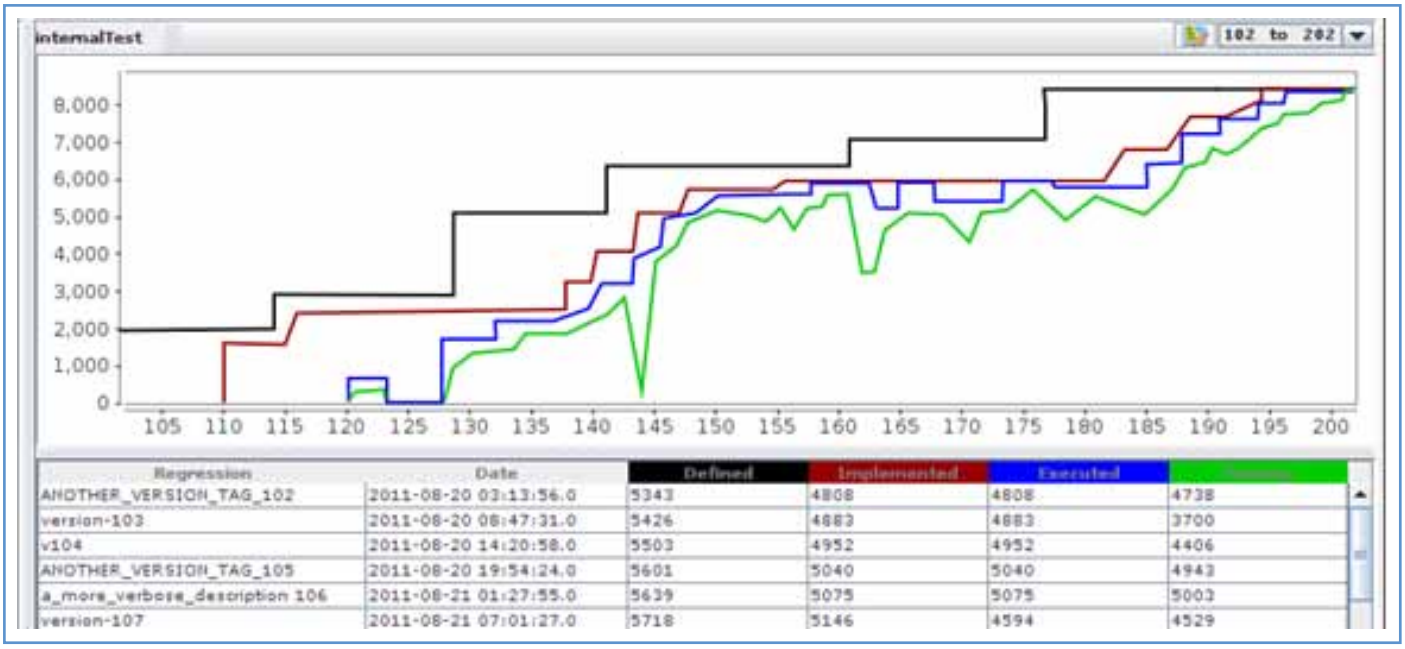
2 *For example, Bull 1998 and Chaos Report 1995.*

Requirements tracing offers a number of advantages. For example, it is possible to more easily understand the impact of a change in the requirements or understand which product features do not actually relate to any requirement. The technique is important in building rigorous software systems and is mandated in a number of industries (U.S. Food and Drug Administration, aeronautics (DO-178B, DO-254), railway transportation (EN-5012x), automotive (ISO26262, IEC 61508) and many others). There are a number of tools (such as Doors, Reqtify, Enterprise Architect, Jira) to support requirements tracing, but our customers reported to us that they did not offer good support for testing. At best they map requirements to tests without any connection to test status, test results and test history. The rest of the article discusses our experiences of automating the collection of such data.

The automation has been embodied in a tool that allows the users to map their requirements directly to tests. As seen above, those requirements are often in the form of a hierarchical tree, and the user can map from any point in that tree to any test through a many-many relationship. This extends the analysis described above to include test orphan analysis (i.e. identify tests which don't actually map to any requirement), and extending impact analysis to include testing. One important advantage is that the user can see what requirement each test maps to. This helps to document the tests: a number of testers often inherit test databases and the first question to answer is "what are they testing?" By mapping them to requirements that question becomes much easier to answer.

The users were keen to ensure that the automation did not just record status (i.e. pass and fail) against a test as this information comes quite late in a project. So the user can record a status of defined (we have identified the tests required for this requirement), written (those tests are now written), executing and passing. Our customers tell us that this gives them a much earlier, more realistic view of the test status than just waiting until tests are running and passing.

The test "executing" and "passing" information is collected through a simple interface that the users call from their existing test automation scripts. The tool also automatically records the source code version of the product being tested. This allows the



Regression	Date	Defined	Implemented	Executed	Passing
ANOTHER_VERSION_TAG_102	2011-08-20 03:13:56.0	5343	4808	4808	4738
version-103	2011-08-20 08:47:31.0	5426	4883	4883	3700
v104	2011-08-20 14:20:58.0	5503	4952	4952	4406
ANOTHER_VERSION_TAG_105	2011-08-20 19:54:24.0	5601	5040	5040	4943
a_more_verbose_description 106	2011-08-21 01:27:55.0	5639	5075	5075	5003
version-107	2011-08-21 07:01:27.0	5718	5146	4594	4529

Figure 2: Tracking test status using "defined", "written", "executing" and "passing?"

user to build up an historic status for the testing which helps the testers to more accurately predict completion. Users are also able to view the test status using the requirements hierarchy thus giving a unique view for each requirement.

The tool has been used in a number of flows and environments. Figure 3 shows how it gets used in a sequential flow (the "V-model" in this case).

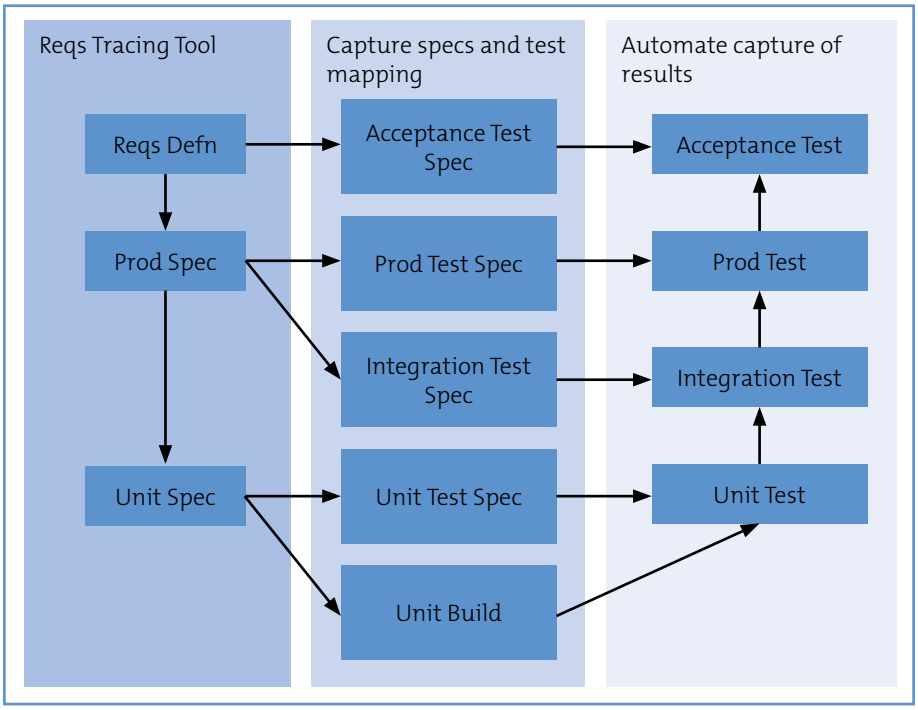


Figure 3: Supporting a sequential flow

However, the automation can also be used to support techniques more usually associated with iterative environments.

- "Use sases" or "user stories" can be recorded as "requirements" and then mapped to their associated tests.
- Test Driven Development: Add the tests into the database and map them to the requirements they test. You can then track how the tests initially fail and then start passing as the product code adds the features they test.
- Testing with scrum: The automation helps to ensure the features being added within the sprint are all mapped to tests and thus increase the likelihood they will get tested within the sprint (which is one of the keys to successful testing in scrum). Also, if tests from previous sprints start to fail, then it is easy to see which features are broken.
- Code refactoring: As you refactor your code, some tests often start failing. The automation allows you to easily identify which requirements or features have been broken.

The concept behind the automation we have described is simple. We record the mapping of requirements (in their varying formats) to tests using a simple SQL database that can also record

the version of the code being tested and the results of the tests from existing automation flows. The advantages for the user are:

- Reports on the percentage of tests defined, written, executing, passing (not just a "pass/fail" status) against the requirements and aggregated for their position in the requirements hierarchy.
- Use the recorded historical perspective for more accurate test completion predictions.
- Support for regulatory-based requirements signoff.

Future features to be added will allow automatic connection to a bug database and code coverage results.

> biography

Dr. Mike Bartley *has been involved in software testing and hardware verification for over 20 years. He started his career in testing of military software and safety-related aerospace applications using formal mathematical methods. He then moved into commercial hardware verification of a 64-bit MPEG4 chip at ST Microelectronics. From there he moved to Verification Manager at Infineon building up a team of over 35 verification engineers using state-of-the-art verification technology to verify numerous chips and design IP ranging from secure chip cards, through automotive applications to mobile phones. Mike then moved to start working with start-up companies in charge of both the testing of software products (tool chain, run-time libraries, applications, etc.) and the verification of the hardware products - firstly at Elixent (now Panasonic) and then ClearSpeed. In these roles he established software testing and hardware verification teams (including offshore resources), flows and processes which were used to sign off numerous hardware/software products and are still in place today.*

Mike gained a PhD in Mathematical Logic from Bristol University. He has since obtained an MSc in Software Engineering and an MBA through the Open University. Mike has had numerous papers published and presented at a number of conferences.

Mike now runs his own software testing and hardware verification company TVS (Test and Verification Solutions) to help companies improve and execute on their software test and hardware verification strategies. Mike has grown TVS to the point where the offices in the UK, India and Germany have a worldwide headcount of over 45 people and a turnover expected to exceed £2M in 2011/2012.

CloudOnomics

by Ian Moyse

Moore's Law back in 1965 predicted silicone power would double every two years. However, what its creator, Gordon E. Moore, couldn't have predicted was the dramatic economies of scale the Cloud would eventually bring to all of our lives. For one, it's helped lead to a drop in price for essentials like computing power and storage by making them more accessible. In addition, it has enabled conveniences no one ever would have imagined four or so decades ago.

There has been a thunderstorm of growing noise surrounding Cloud Computing in the past 24 months. Vendors, analysts, journalists and membership groups have all rushed to cover the cloud medium – although everyone seems to have their own opinion and differing definition of Cloud computing. According to the most common definition, it is Internet-based computing where shared resources, software and information are supplied to users on demand, rather like a utility company would supply electricity, water or gas. The term is not new; vendors such as Salesforce.com have provided Cloud services in different guises for many years. Other players have been swift to get on board, including Microsoft, HP, IBM, Amazon and Google, to name but a few. Put simply, users now have the choice of a new way to consume computing power, applications and data. It is no longer necessary to buy software on a floppy disk or a CD. Instead, you can have immediacy of delivery through the Internet for an application you want now. Users have been educated into this way of working with iTunes and app stores, and they've come to expect a seamless link between their locally run application and data and information from the Internet – and at a very digestible and economic price point. Buying a robust, polished application or game for below £1 is now taken for granted.

As an average user you are also likely to be using Cloud computing in the form of webmail, Flickr, YouTube, Facebook and a plethora of other services; storing what you would consider private information in the Cloud without knowing where it is in reality... or even caring. In effect, Cloud has become a very simple and trendy way of describing all things that occur outside the firewall, whether it be on a corporate network or on your home PC. And Cloud computing is already helping to shape the way we digest IT both at home and in the workplace. It is simply a new 'form factor', a new way of delivering a solution to a customer. We have seen new form factors disrupting and changing many market sectors already. Think of Blockbuster Video, once the darling of the entertainment world and now struggling to survive against the new delivery factors of Netflix and LOVEFiLM. Tower Records, once a worldwide brand, has been put out of business by the ability for users to now purchase music faster and cheaper via iTunes and online music stores. The same trends are occurring in computing.

Today we're able to use a mobile device with massive power and local storage to locate and download from virtually anywhere in the world an application for as little as 59 pence. Think for example of Shazam, which identifies songs you can't quite discern after it listens for just a few seconds. Leveraging its Cloud database, Shazam also lets you buy and download the song via your smartphone. All of this – the convenience, the low cost, the power on the local device – is driven by the Cloud. Customers do not choose to use it because it's Cloud, in fact most would not even think of it this way or even care how it works, it simply does and does its job well, simply and cheaply.

The Cloud has not only driven down costs, but it's helped increased our satisfaction with – and expectations of – our Internet experience. It's enabled mobility and delivered immense computing power to anyone, anywhere at any time. The Cloud has also driven the success of many vendors and will continue to do so as developers deliver applications that are faster to market and reach a wider commercial audience at a lower cost of delivery.

Cloud computing offers substantial benefits including efficiencies, innovation acceleration, cost savings and greater computing power. No more 12-18 month upgrade cycles; huge IT burdens such as system or software updates are delivered automatically with Cloud computing and both consumers, small and large organizations can now afford to get access to cutting-edge innovative solutions. Cloud computing also brings green benefits such as reducing carbon footprint and promoting sustainability by utilizing computing power more efficiently.

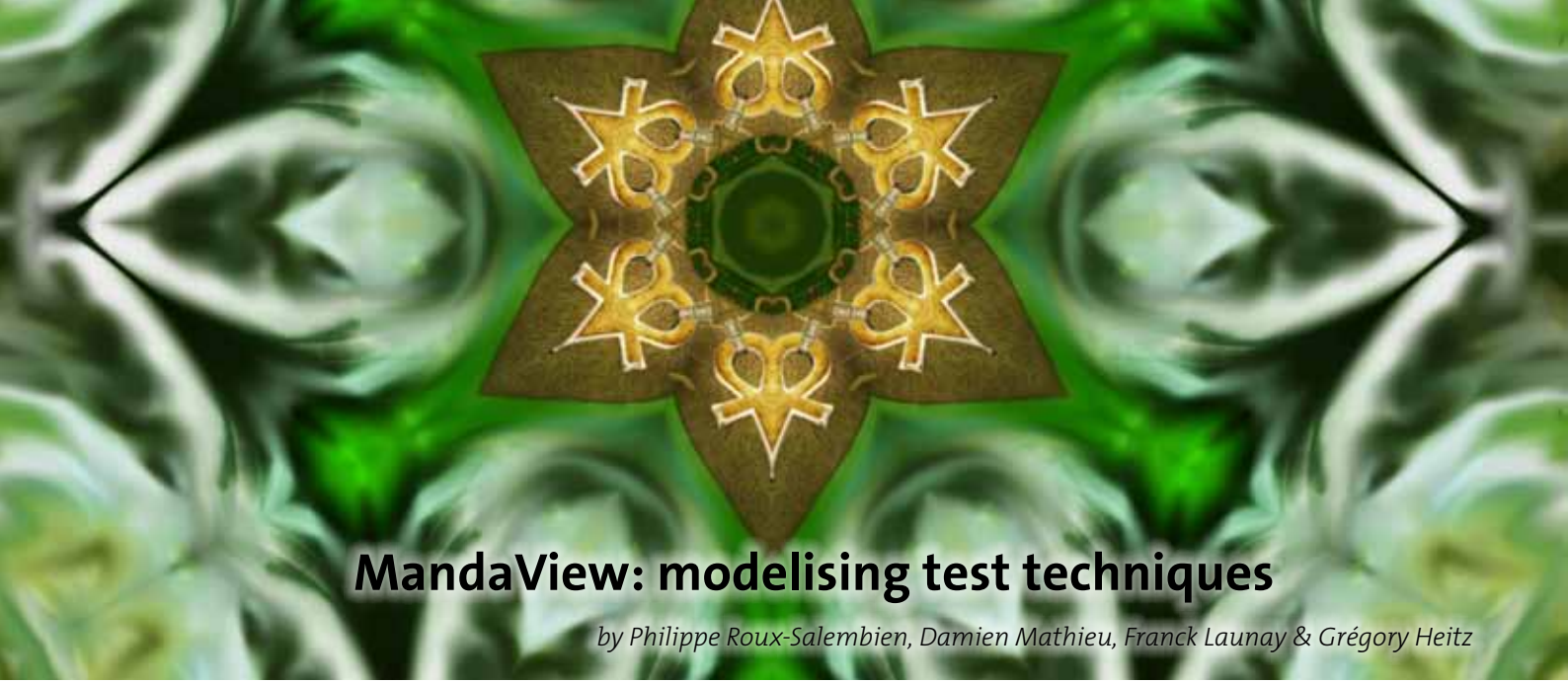
We should expect to see more changes in the size and delivery methods of the technologies we use – where very small files, programs or devices connect to the Cloud where all of the benefits are stored. Such client/cloud configurations are a boon for consumerization as our appetites for an always-connected "iWant" lifestyle increase.

In ten years on iPhone 14 and iPad 11, will we see applications that are free and pay 1p per use perhaps? Or will we see others employing new models that yet again change the way we digest and pay for computing power and information?

More changes to the Cloud economics that we won't see coming are inevitable. Perhaps an update to Moore's Law will be formed to hypothesize that the number of applications running in the Cloud will double every two years; based on today's adoption and consumption rates, however, we're more likely to see this being every two months.

> biography

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



MandaView: modelising test techniques

by Philippe Roux-Salembien, Damien Mathieu, Franck Launay & Grégory Heitz

We have all attended training courses, presentations and seminars, and have seen a wide variety of schematics and graphics used to illustrate ideas and concepts. Do we really remember them and do they really help us to learn and understand?

From a pedagogical point of view, this is a real concern. Illustrations are far more effective than lengthy texts to emphasize ideas, but they can become less effective when there are too many.

With this in mind, we have devised a new concept: MandaViews (MDVs). MDVs provide a family of models for demonstrating test techniques. The goals are on the one hand to provide pedagogical ideograms to facilitate the understanding and long term memorizing of all of the different categories of test techniques (black-box, defect and experience-based and white box), and on the other hand to provide a meaningful project reporting tool for managers.

Why suggest a model for test techniques modelization?

A pedagogic need

Many of the different techniques are not only well understood but also memorized and organized by testers who want to improve their skills. To achieve a good and effective understanding can involve a considerable amount of work: compiling paperwork, creating checklists etc....

A documentation approach

Furthermore, providing an overview of all the different test techniques actually used in each step or level of a testing project could provide valuable feedback to the stakeholders on the range and the complementarity of the panel of test techniques currently in use.

Mandalas: from ancient tradition to applied pedagogy

In Sanskrit, mandala means circle, and these concentric diagrams are commonly used as a meditation tool.

For several decades now, this ancient concept has been adapted and used by teachers to illustrate related concepts and notions, making them more understandable[1].

Philippe Roux-Salembien attended a conference on 'How can we increase our effectiveness in helping our children do their homework', held by senior teacher Armelle Géninet at his daughter's secondary school. Mme Géninet explained, among different concepts and ideas, how the philosopher and educator Antoine de la Garanderie had developed the 'Mental Management' idea and its 5 key elements to describe the entire learning process: Attention, Memorization, Comprehension, Reflexion and Imagination[2]. This research identified the need for the learner to 'spatialize knowledge'. The mandalas described above are used to assist in assimilating knowledge in fields as diverse as mathematics and English. The specific use of mandalas in such pedagogic applications compared to other graphic models, relies on the use of symmetries, graphical links and codes between the different items. The combination of these enhances spatialization, understanding and memorization of the concept to be learned.

During this conference, Philippe had the idea to apply those pedagogic mandalas to his own particular field, of testing, and more specifically, use it to represent test techniques. This idea was supported by the variety of available test techniques in each category as described in the ISTQB syllabi. He therefore designed a first model for black box and experience based techniques and named this family of mandalas dedicated to testing: MandaView. Some time later at a meeting with Damien Mathieu, Franck Launay and Grégory Heitz, Philippe discussed this concept with them. They agreed on the originality of the idea and discussed the potential of MDVs in testing-related activities. From there they decided to continue exploring other uses for MDVs that could be shared with the international tester community. This is how the idea of this article came about.

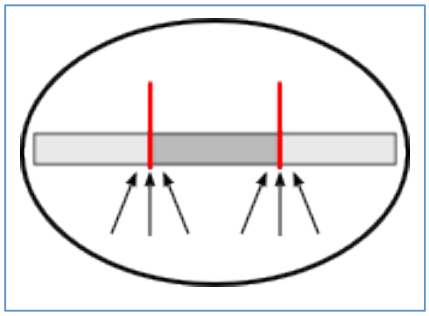
1 *'Graphismes et mandalas d'apprentissage', CP, A. Géninet, éd RETZ*

2 *'Les profils pédagogiques', A. de la Garanderie, éd Bayard*

MandaView basics

Like mandalas, MDVs are based on circular diagrams. These circles are divided into sections to represent each item. Each section contains a graphic symbol (ideogram) illustrating the test technique that it represents.

The first principle of understanding MDVs is the clear association between the ideogram and the principle of the related test technique.



Example: Ideogram for boundary value analysis

Thus, the process of remembering the core principle of each test technique on a long term basis, even those not often used, is made easier.

Positioning and grouping the techniques according to their affinities and categories also helps to identify and understand their potential daily use.

MandaView Model for effective pedagogy

The three aims of the MDV Model rely on the process shown in Fig. 1.

Firstly, spatialization enables a global approach for all the techniques within a category by grouping related techniques. Secondly, visualization is improved by means of color themes and ideograms to help understanding and retaining the principles of the particular techniques concerned and their relationships. Thirdly, the resultant unified models improve the ability to memorize the information long-term, once some time has been spent initially to understand the entire model.

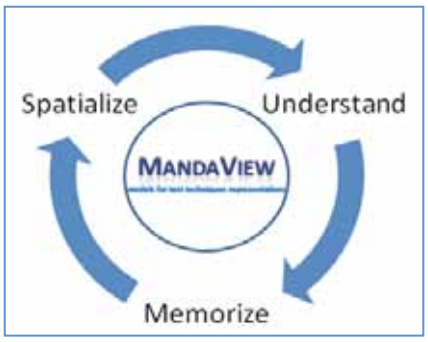


Figure 1: Key features of MDV Model, inspired by Antoine de la Garanderie research

MandaView Model for modelization of specification, defect and experience-based techniques

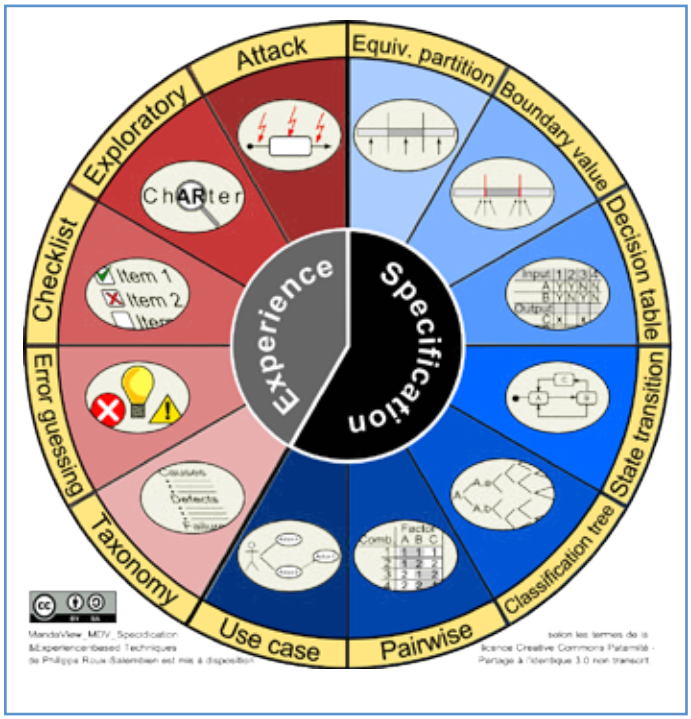


Fig.2: First MDV: example for specification-based techniques and defect and experience-based techniques.

Why grouping black box and experience-based?

Black box and experience-based techniques together probably constitute the test techniques most commonly used by test analysts, therefore it makes sense to group them together in a single diagram. Furthermore, they encompass around a dozen different techniques in accordance with ISTQB, which is ideal for the construction of the first MDV.

The circular shape, which mandala represents and is therefore the foundation of MDVs, enables a quick overview of the test techniques being demonstrated. Specific groupings of related techniques, with associated comments, are thereby possible. Furthermore, dividing the MDV into three areas eases knowledge spatialization.

1. The test techniques' names in the external ring;
2. The ideograms designed to represent the basis of the related technique, in the main ring;
3. The central zone showing the categories within this particular MDV.

The central zone of the MDV shows the limits between the two categories: specification-based on the one hand, and defect and experience-based on the other. These two parts of the picture are also identified by two main background colors: each specification-based technique is in a specific shade of blue; whereas each experience-based technique is in a red shade.

The ideograms relating to the test techniques are easy to insert into the MDVs and provide assistance for understanding and memorization of the core principles of the techniques.

The first two ideograms of specification-based techniques show the key roles of the equivalent classes within them. Then the ideogram of a decision table technique shows an example of a decision table which is most commonly used and easier to identify than a cause-effect graph.

The state-transition ideogram shows the starting point with three different states, each of which relates to the others. The classification tree shows the outline of an example of this tree, whereas pairwise testing is illustrated by showing only a part of the particular table. The use case ideogram displays a simplified view of the usual UML diagrams.

On the defect and experience-based side, we chose to highlight causes, defects and failures in the taxonomy testing ideogram according to the ISTQB definition. Error guessing testing is probably fairly simple to recognize, as is for checklist testing. Exploratory testing shows a magnifying glass zooming on the word "Charter" illustrating the concept of using tester's intuition around predetermined themes. And finally, the software attack ideogram shows lightning strikes, representing the attempts to force failures through interactions with the software and its environment[3][4].

One can observe that the spatialization of the twelve techniques of the two categories enables grouping, which helps both understanding and memorization. Two examples: on the one hand equivalence partitioning testing and boundary value analysis testing are put to one side as they are both based on class equivalence; on the other hand, software attack might use the other experience based techniques which is why it has been put on the last position of these latter techniques.

MandaView Model for modelization of structure-based techniques

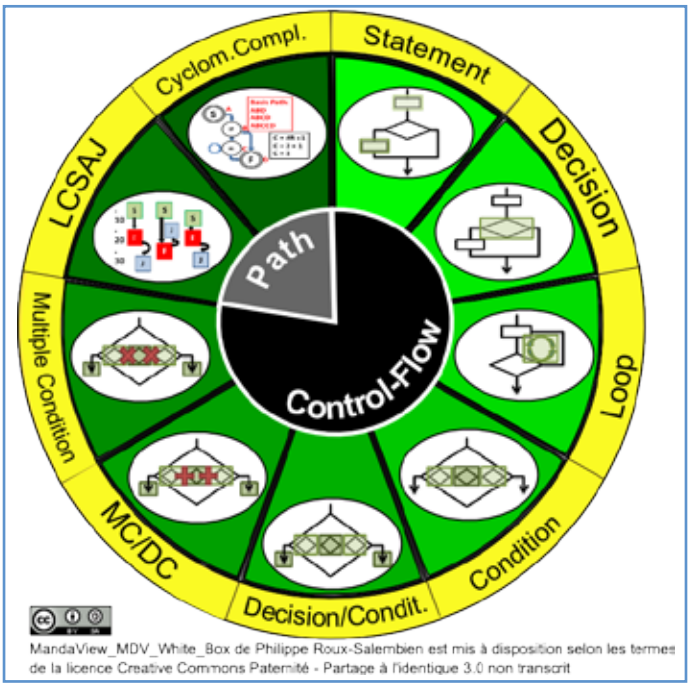


Fig.3: MDV for structure-based techniques

Two main structure-based (white box) techniques: Control Flow Testing and Path Testing

In order to illustrate how useful white box techniques are for testing, we will quote Black & Mitchell[5], whose book was very helpful for writing this chapter (together of course with the above mentioned ISTQB Advanced Level Syllabus): for a 'world class, excellent system test team using all black-box and experience-based techniques', doing all the job the best way, 'at the end of that, have they done enough testing?'... 'May be as much as 70 percent of all of the code in the system might never have been executed once! Not once!'. Please note that all technical definitions used in this chapter are quotes from this book.

We will now focus here on the reasons why we conceived the 7 ideograms for control-flow techniques and the two ideograms for path testing, which were used to constitute the White Box testing MDV. Of course, the structure of the second MDV is similar to the first one.

With the aim to design the different white box techniques ideograms, we chose graphical representations inspired by control-flow graphs, but somehow different in order to concentrate the essentials of each technique in each of them.

As defined in the ISTQB glossary of testing terms, statement testing is a white box test design technique in which test cases are designed to execute some or all of the available statements. The selection of the statements to be tested is based on their being executable. Among all control flow techniques, this technique offers least code coverage, and it is the first ideogram in the White-Box MDV. The black arrow means a set of consecutive statements and the pale green box represents the tests in this part of the code.

The second strongest level of structural coverage is the decision – or branch – coverage. It looks for decision, often 'IF' conditions or 'SWITCH' conditions (which can be seen as a complex set of atomic 'IF'), whereas 'DO...WHILE' or 'WHILE' are loop decisions. As the decision expression is tested in decision coverage, we represent it as a green box, with two emerging arrows related to the TRUE and FALSE outcomes of it.

We included loop coverage in the White-box MDV, despite it not being presented in the ISTQB Advanced Syllabus. The chosen way to represent loop coverage is rather straight: an arrow, covered by statement coverage (pale green box) with aside a loop with a deep green box representing the tests to be designed. Let's note that this technique is very hard to apply extensively, and that the above mentioned Boundary Value Analysis technique is then helpful. This shows rather accurately how black and white box techniques can be and even have to be related.

The fourth structure-based technique is condition coverage. As decision coverage checks if we have exercised each decision both ways, condition coverage discusses about how the decision is made. That means that 'when a decision is made by a complex expression that eventually evaluates to TRUE or FALSE, we want to make sure that each atomic condition is tested both ways, TRUE and FALSE.'. We therefore designed the condition coverage with a decision fork, and inside the decision, some atomic conditions, each to be tested. Something very important in this technique is to note that condition coverage does not automatically guarantee decision coverage.

3 *Advanced Syllabus, ISTQB, Oct. 2007*

4 *Advanced Software Testing Vol.1, Rex Black, 2009, Rocky Nook Inc.*

5 *Advanced Software Testing Vol.3, Rex Black & Jamie L. Mitchell, 2011, Rocky Nook Inc.*

Thereafter comes a stronger coverage with decision/condition coverage, also called condition determination coverage. As it combines both previous coverages: each atomic condition and overall expression are to be tested both ways, we assembled both source ideograms in the resulting one, with a decision showed by the deep green box, including the atomic conditions represented by triangles.

After this, Modified Condition/Decision Coverage (MC/DC) is yet a stronger level of coverage. It adds one more factor: 'Each condition must affect the outcome decision independently while the other atomic conditions are held fixed'. As the number of tests to be designed for covering MC/DC is close to n+1 (with n atomic conditions), we separated the latter by a red '+'.

The last of all of the control-flow coverage schemes, which provides the highest coverage, is multiple condition coverage, where we test every possible combination of atomic conditions in a decision. As the number of tests is maximum 2n, before eventual short-circuiting, the chosen graphical relation between atomic conditions is a dark red 'X', for multiplying combinations.

Now the two last structure-based techniques are not based on control-flow coverage, but are related to path testing.

As explained by the ISTQB Glossary, the first path testing technique, the LCSAJ (Linear Code Sequence And Jump), consists of the following three items (conventionally identified by line numbers in the source code): the Start of the linear sequence of executable statements, the end (or Finish) of the linear sequence, and the target line (or Jump to) to which control flow is transferred at the end of the linear sequence. The ideogram for LCSAJ shows then: line numbers, S for starts, F for finishing and J for jumps. Two LCSAJ tests are exemplified.

The cyclomatic complexity, developed in 1976 by Thomas McCabe, helps us evaluating code complexity and the amount of test required to cover it. Each test is a unique independent path through the module – with no iterations loops allowed.' Three different basis paths are possible in the ideogram. As the number of basis paths is equal to the cyclomatic complexity, we highlighted in the ideogram the calculated cyclomatic complexity (in this case Cc = 2 'enclosed Regions' + 1 = 3).

Conclusion

MDV offers a new approach for the demonstration of test techniques. Firstly, the MDV Model provides a useful graphic way to spatialize, understand and memorize each test technique thanks to the specific feature of mandalas. Secondly the MDV Project could provide a practical and all-encompassing demonstration of the test techniques used within a single project.

MDVs are still under development (i.e. for project samples or reviews) and we would be pleased to share our models with you and to receive any feedback. Designing meaningful ideograms is quite a challenge for some techniques, so any complementary suggestions on these would also be most welcome.

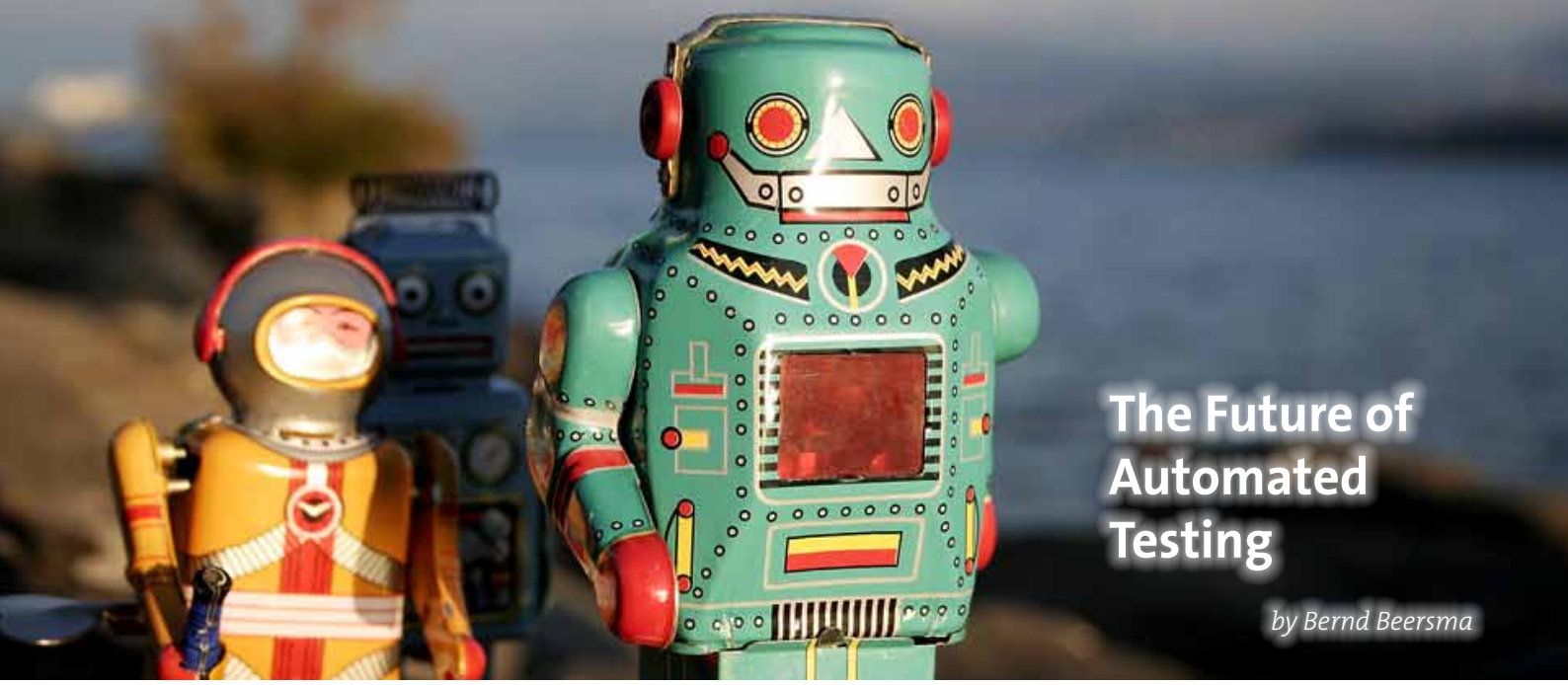
Note: Special thanks to Mme Géninet for inspiring in me the MDV concept with her mandalas and having kindly allowed to reference her work in this article. Thanks also to Thierry Charles for his help in the first draft paper drawing of the model I had conceived, and thanks to Eileen Basgallop for her very kind help in correcting the English in this article.

And of course, special thanks to Damien, Franck and Grégory for all their support and significant help in challenging, improving and developing for tomorrow both MDV concept and this article.

> biography

Philippe Roux-Salembien
He has been involved in the services industry as developer, test support, test lead and test manager for 12 years.
He has been engaged mainly in IT services for clients in banking, telecom and other industries. His involvement has primarily been in functional testing, having gained experience in TMAP NEXT®, also in training and support for some commercial tools. He has a French equivalent to a MSc. in computer science, and is an engineer in biotechnology in France, Switzerland and Germany. He is an ISTQB Certified Professional, and is interested in returning to an international role in the near future.

Damien Mathieu, Franck Launay and Grégory Heitz
The co-founders of FACTory Consulting are ISTQB Certified professionals. Damien and Franck are former network and system technicians. They've later come to software testing and now mainly concentrate on test analysts' tasks, training and testing tools usage and best practices. Gregory has worked for about 10 years as test manager and trainer in various companies (electronic banking, pharmaceutical and multimedia).



The Future of Automated Testing

by Bernd Beersma

The future of testing? Where will we go with testing? Since my main focus is on test automation, this article will describe my vision on the future of automated testing.

To understand where the future in automated testing will be, we first have to understand where it was. So let's go back to the future...

We will take a walk down memory lane, and we will do so following my experiences with test automation from 1999 until now.

First Contact

I first came in contact with testing and test automation during an internship at a hard and software production company in 1999. I was on this project for creating a web based interface for configuring a Sun Solaris system which was the beating heart of a complex hardware solution. Developed in Perl I created this interface, but being a software engineer I believed that everything I created was working fine. One day during lunch I talked to two of the Java developers and they told me I should test my just developed software, so I could verify if it worked as requested. They introduced me to a software testing method from The American Department of Defense (DOD) and showed me how I could use this to test my software. Ok, now, I wouldn't think about testing my own software, but at that moment it was the best I could do, because there were no testers in the company.

So I tested and tested my software, setting up a test plan, creating test cases etc. and of course found some defects. During another lunch I asked the Java developers how they use testing as part of their development process. They said they had to run thousands and thousands of tests for their embedded software, but they did it automatically. They programmed their test cases within the embedded software and executed these test cases time and time again, guaranteeing the software worked correct and as requested. This was my first contact with automated testing, automated unit testing.

After that internship I decided that testing would be a part of my life. Finishing my BA in Software Engineering I always used the DOD standard to test my software. Although it still was manually, it was more testing than most of my fellow students did.

A Silver Bullet?

In 2001 I started at the Computer Management Group (CMG) in the Netherlands as a software tester, adopting the TestFrame™ method as the method for designing test cases. During my first weeks at CMG I followed courses on test case design and, automated testing with a record and playback tool from a well-known tool vendor. I learned how to use the record and playback tool to automate test cases. After that first encounter I rapidly got acquainted with other automated testing tools in the field of record and playback. All of you that were interested can figure out which tools I used at that moment. Those tools were similar to each other and could create test cases in a quick and easy to understand way, generating a large number of test cases. But was this the silver bullet? No it wasn't, and the future was not even close at this moment.

The First Generation, Record and Playback

The so called first generation of automated test tools consisted of record and playback tools. This basically worked like a VCR (ok, for our younger professionals a DVD or HDD recorder). A tester needs to record his or her actions on the system under test (SUT). The recording tool will record this and create an automated test script of those actions. After this session, the tester can play back this just created script and the tool will interact with the SUT simulating the tester and the actions of this tester.

The biggest problem was the maintenance issue. Because of the static scripts created with the record and playback tools, changes in the system under test or even test data challenged the test automation engineers. This meant changing the automated test scripts which is okay for one script, but what if the login sequence for your thousand plus test cases needed changing, or new objects in your SUT are created and they are mandatory? At that moment you needed to adapt your automated test scripts to these new situations, creating a lot of work for the test automation engineers.

So, what was the future for us, the test automation engineers? Because this time consuming test automation certainly was not the best solution for test automation.

The Second Generation, Data Driven Test Automation

As I learned in my next assignment for CMG at a government organization in the Netherlands, the solution was data driven test automation. This basically meant that you take your test data out of your automated test scripts and place it outside your tool. This could be an Excel, a CSV or even a plain text file. Storing your test data outside of your tool meant easier maintenance on your test data and faster creation of additional test cases. For the test automation engineers this meant less maintenance on test data, but still the maintenance problem was the biggest challenge for successful test automation. That's why the data driven approach wasn't the future of test automation, we were getting ahead, but still had a long way to go.

The Second Generation, the Reuse of Generic Functions

In addition to this data driven approach, the logical next step was to create functions of repeatable or generic parts of your test automation software. The project I worked on did both and it was based on the action or keyword principle of TestFrame.

Let's take a closer look at creating generic functions as part of test automation. I mentioned earlier the login sequence needed in a lot of test scripts. This login function is a good example of a test step which can easily be transformed to a generic function. If you transform this recorded login sequence into this function *(Function Login(User, Password))*, it means that you can use this newly created function everywhere a login is needed, just refer to *Login(Bernd, Bernd77)* and it works with the credentials you send to the function. Slashing maintenance, because if something changes in this function, we only need to change the function and the login will work again for all the test steps where it is used.

Again, one step closer to the future, but we aren't there yet.

The Second Generation, the Use of Action Words

Back to the action words: An action word describes an action on your system like "Enter field Name" in combination with the value "Bernd" or "Press button" with the value "Enter". These action words correspond with a function in your automated test script library and are executed on your SUT.

So these generic functions of action words soon led to a test automation framework which was used to create automated test scripts more easily. The testers would create their test cases in Excel and the test automation engineers registered those action words in their framework; this resulted in a big decrease in maintenance. However, this still wasn't ideal and we needed to improve our framework. This resulted in so called functional action words. Again one step closer to the future, but still not there.

The Second Generation, the Use of Functional Action Words

The functional action words were action words which consisted of a static part "Fill Screen Personal Information" combined with all arguments used to fill this screen like "name" and "address" with the corresponding input values and often an action to perform "press next". So now a tester did not have to use multiple action words to fill in a screen, but could use the action word "Fill Screen XXX". Again this led to a decrease in maintenance and test cases were also easier to create. If we speak of generations of test automation, I believe this was the second generation of test automation, record and develop test automation. However, we still use the same tools as before, only we use them in another way, so this is still not the future of test automation.

The Second Generation, a Generic Technology Independent Test Automation Framework

After leaving this project I was convinced this was **the** approach for test automation. And so I used this framework and approach for many years at several projects and customers until I started a project at a big insurance company.

Becoming the lead automation engineer I was responsible for setting up test automation on a large scale and I was convicted we (me and my colleagues) could improve the framework we had been using for years and years. This led me to introducing an approach which I described in my previous article in Testing Experience 14, *Improving automated regression testing*.

This approach is based on Generic Test Automation and Generic Test Case Design and is technology independent. This basically means that it doesn't matter which system or application you want to test. Interacting with the SUT is always the same, because it doesn't matter if the screen you need to fill out is an HTML application or an SAP screen. If we look at the business logic, the path we need to test is described as:

A. Search for the book "Implementing Automated Software Testing" by Elfriede Dustin (Screen A)
B. Place the book in your shopping cart (Screen B)
C. Go to the check out (Screen C)
D. Review your order (Screen D)
E. Confirm your order (Screen D)

So these are five steps on a business level, but only four steps on a technical level, screen A until screen D, but more importantly, they are all screens; screens with different objects, different checks and different actions, but still screens.

As we have been building generic functions for objects like an HTMLEditbox or an SAPTextField and putting them into libraries for years, the next step in test automation took shape. Because the only difference between HTMLEditBox "Name" and HTMLEditBox "Address" is the technical information needed to steer this object in the SUT. So our libraries where full of functions to steer different applications and environments. The only thing that was left to do was to identify the technical information and put this in an external file, in our case an XML File. This could easily be done by the automated testing tools we used at that time, commercial tools from big vendors or open source tools. For a further detailed description of this framework, I would like to refer to my previous article.

I believe this Generic Test Automation framework is a good approach for test automation, and stands between the second generation of test automation and the future. It brings back maintenance to a minimum and lets testers create test cases in a logical and easy to understand way, but still, you need test automation engineers with technical knowledge and programming skills. And these are not skills every tester possesses of even wants to possess.

The Next Generation

This leads us to the future of test automation and what does it look like? So I have been around test automation for several years, I worked for a large Dutch test company, a worldwide tool vendor and now a smaller Dutch test company. I had my share of tools and still come across new tools very often. So where is the future?

I think it's in different areas like scriptless automation, user friendliness, standardized integration between tools and the mindset of combining different tools to fit your needs.

First of all, I see new, upcoming and promising tool vendors with an innovative approach on test automation. The new approach is scriptless automation, so functional testers can set up automated test cases without the technical knowledge or the need for programming skills. Tools that are born from the need to create an approach by which testers can setup test automation in a simple and logical way. I like to use the phrase "Created by testers for testers". And I believe there will be more and more vendors who will try to go this way.

Another thing I hear very often is that automated test tools are too complex. The majority of testers would like a test tool that is easy to use, adaptive and comes with a low learning curve, so they won't need months of tool training to get started using this tool. So in the future we need tools that have these characteristics.

Furthermore, what the future of test automation needs is standardized tool integration. Why is it that every vendor creates its own interface for test cases, defects, requirements etc.? It would be easier if we could use *tool A* for requirement management, *tool B* for automated test execution and *tool C* for defect management because these are the best tools for the job. Also when we decide to use another tool, it's a pity we need to migrate our automated test scripts, because they cannot be used by this new tool. Ok, I must admit a lot of closed source products do interact on a minor scale with each other. One vendor has developed a platform on which we can use closed source, open source or even homebrew. But still, there is a long way to go.

Finally, the future of test automation needs the mindset to use multiple test tools and not wait for the silver bullet, because it won't come. I came across a lot of projects and customers who still think there is one single test automation solution to solve all their problems. I am sorry, but there simply isn't. I believe you need to choose the tool which best fits your problem or needs. If you are a web developer and need to run an automated test each daily build, maybe an open source webtest tool is the tool for you. If you are a functional tester on an SAP system, you may want to set up your automated regression test with a commercial tool. This is not only applicable for test automation tools, but also for other tools which support your test process like defect management or requirement management.

Looking back, we came far from the early days of test automation, but there is still a long way to go to use test automation on a wide scale. And we as testers take part in creating this way, together with tool vendors, open source communities, and automation enthusiasts.

So, my final conclusion on the future of test automation:

The future of test automation is now, and you're in it...

> biography

Bernd Beersma
is competence leader test automation and senior test consultant with Squerist. He has over 9 years experience with different forms of test automation and performance testing for different companies.
Bernd holds a bachelor degree in software engineering and became acquainted with software testing during this period.
During his numerous customer assignments Bernd created different frameworks and solutions for test automation with a broad variety of tools. These different approaches led to creating a generic approach for test automation.
As a result of his knowledge about test automation, he also gives advice on how to implement test automation and does workshops and trainings.
Bernd is a member of the TestNet Test Automation workgroup and co-initiator of the Test Automation Day.
His goal is to keep on learning and thus improving his skills on test automation.

Online Training

Díaz Hilterscheid

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

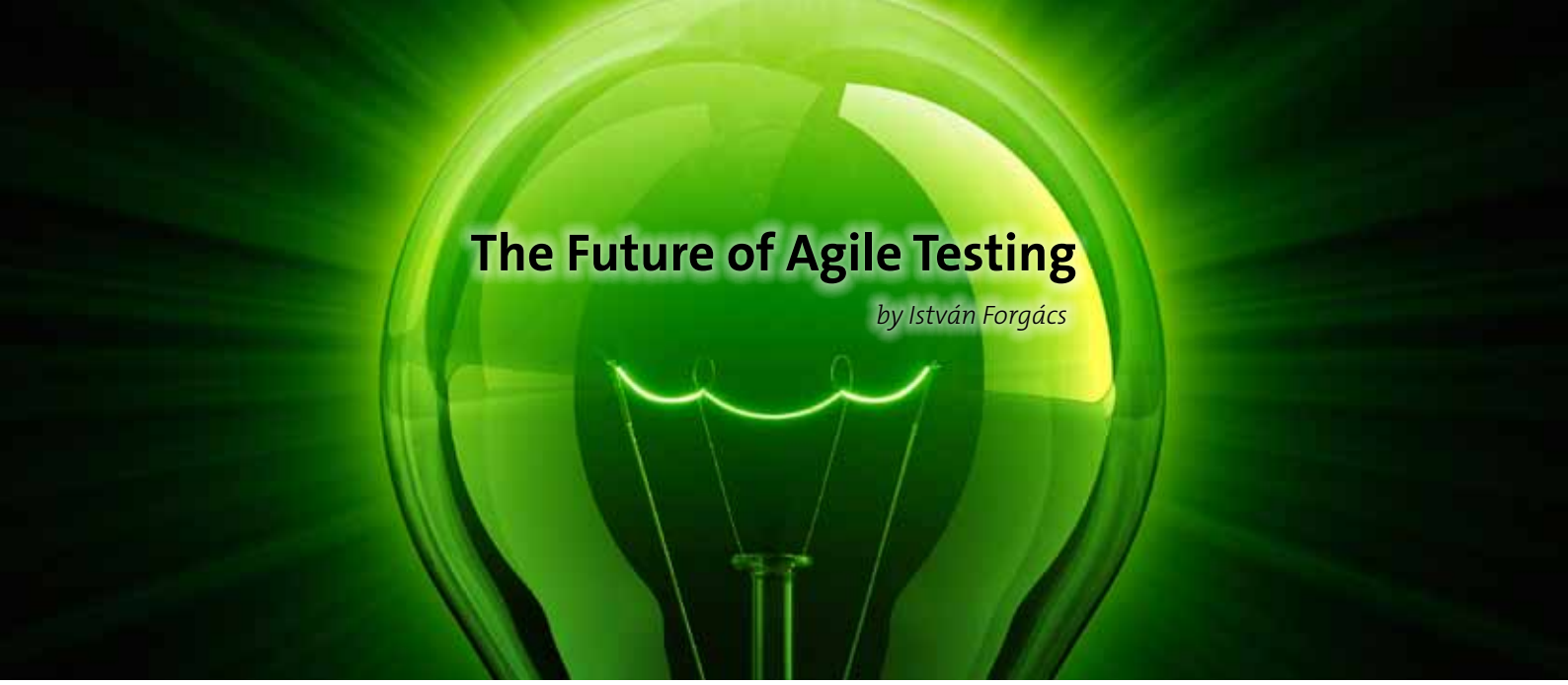
iseb Qualifications from BCS

ISTQB® Accredited Training Provider

ISTQB® International Software Testing Qualifications Board

Díaz Hilterscheid

te testing experience



The Future of Agile Testing

by István Forgács

In the past ten years there has been significant progress in agile testing. A number of new methods have been introduced and dozens of related testing tools have been released. Nowadays agile testing is a strong and essential element of agile development. However, there is no unified „theory" of agile testing, the key methods are considered in isolation and agile testing tools are not matured enough. In this article, I will try to predict the possible direction of agile testing knowing, very well though that nobody knows the real future.

The current situation

To predict the future we should start from the present. My vision of the future is restricted to only those methods and tools which I believe to be the most relevant in this context.

Executing specification

One of the key progresses of agile development is that **specifications are actually code and thus executable**. Instead of creating a comprehensive specification, agile starts from user stories, where a user story covers a required feature, for example a user login. This can be considered a piece of specification. The *agile team* should express this specification by examples called *scenarios*. The examples should fully cover all aspects of the user stories. Each example is related to a test which can be expressed by code applying a very simple language called Gherkin [1]. The syntax of the line-oriented Gherkin language is very simple and contains keywords such as *Feature, Scenario, Given, When, Then, And*.

Given describes the initial state or precondition which should be set for the test. *When* describes the action to do with the actual input. *Then* describes the expected outcome, i.e. the output. Since all three may contain more elements (initial states, inputs or outputs), these are connected by the keyword *And*.

A small example of this executable specification is shown in Figure 1. It was generated by applying the Java tool Concordion [2]. For simplicity it contains only two of the necessary scenarios. We can see that this specification appears as simple text that is understandable for everybody. The original HTML file (see later in this article) contains additional information which is hidden when displayed by a browser.

Feature LogIn

User story: login somebody who already has an account.

Scenario happy path

Given a user with login name **"Smiths"** and a password **"Shtims2011"**.
When user is logging in as **"Smiths"** and enter password **"Shtims2011"**,
Then the message **"Signed in"** appears,
And the user is logged in.

Scenario wrong password

Given a user with login name **"Smiths"** and a password **"Shtims2011"**.
When user is logging in as **"Smiths"** and enter password **"Shtims2010"**,
Then the message **"Wrong login name or password"** appears,
And the user is not logged in.

Figure 1. Example of an executable specification

The first scenario contains the „happy path" where the login is successful. The second one covers the case of a wrong password. (I omitted a third scenario where the login name is wrong.) .

In this article we refer to this sort of specification as an *executable specification*. The advantage of this method is that the executable specification and the related code will match for the whole project life cycle, it's easy to construct and understandable for everybody. Since specification/test description precedes coding, this is a test-first method. This is also called Acceptance Test Driven Development (ATDD) [3].

Test Driven Development (TDD) [4] is a code-based unit testing approach where tests are also created first, prior to code implementation. TDD is made by the developer (instead of a team) based on his/her knowledge of what should be tested when the code will be ready.

D. North realized then that an expressive test name is helpful when a test fails, for example a test method such as **testShouldFailForMissingAccount** is more understandable than for example **testMissingA**. He called his method as Behavior Driven Development (BDD) [5].

ATDD and TDD demonstrably reduce the number of defects by 40-90% [6]. I assume this is because the team/developer first concentrates on WHAT to do and only then on HOW to do it. In this way, more elaborated code can be implemented.

Continuous testing (CT)

This is the „stepchild" of agile testing. CT has been introduced by D. Saff and M. D. Ernst in their paper „Reducing wasted development time via continuous testing" [7]. They argued that similarly to continuous compilation of Integrated Development Environments (IDE), test execution is necessary after each save of code changes. The rationale behind CT is to detect any defects immediately after the code has been modified. The cost of fixing bugs immediately is much lower than even one working day later, especially if the team introduces several modifications. For CT users if one of the team members makes an erroneous modification which corrupts somebody else's code, then the defect is revealed immediately.

Theory of agile testing

In the famous and often referenced paper of A. Bertolino titled *„Software Testing Research: Achievements, Challenges, Dreams"* [8], she outlined a dream of a universal theory of testing, meaning *„one coherent and rigorous framework to which testers can refer to understand the relative strengths and limitations of existing test techniques, and to be guided in selecting the most adequate one, or mix thereof, given the present conditions."* Though there are excellent books on agile testing [3,9], these are for practitioners with good case studies and stories. In fact, testers should base their testing exclusively on case studies which are probabilistic, and in some cases comparing them gives us very different results. Though I don't predict that anybody could find „the final universal theory" of agile testing, researchers on software testing will surely move to this promising area. Based on this, I hope that agile test methods and processes will be unified, more advance testing tools will be released and more relevant case studies can be conducted.

An example where research could help is the common consideration of Test Driven Development and Acceptance Test Driven Development. Both methods are discussed in isolation, though their use in parallel is one of the most promising directions of software testing or even for software design.

The future of Continuous Testing

CT only means that tests are executed after each code snippet has been changed and saved. Really, in the first implementations of CT all test cases are executed. This can be a very time consuming process which prohibits even the immediate error detection. This is the reason why „first generation" CT tools were not successful. Some more sophisticated tools are based on test selection. Instead of running all the tests, it is enough to select only the ones which may fail. These are the new tests and the ones which are affected by a code change, i.e. for which the test execution covers the code part that has been modified. Current tools, however, still select and execute lots of unnecessary test cases.

Future CT tools will select only those tests which are really necessary to be executed. To do this, we should know the execution trace (history) of all the test cases and which part of the code has been changed. Then a comparison of these information results in the tests that may fail when executing again. For example, if I modified the *then* branch of an *if-then-else* statement and my test also covers the *then* branch, the test is to be executed. On the contrary, if my test covers the else branch, then the test cannot fail since the execution traces only unmodified code part. Therefore, this latter test doesn't need to be executed.

In future, CT tools will be able to collaborate with different ATDD and BDD tools entirely, which is not the case at present.

Continuous Coverage (CC)

If a continuous testing tool works at a branch level, while modifying the code results in no tests to be executed, it means that the modified part is uncovered. Therefore CT tools at this level of precision can be extended to become a continuous coverage tool. The only thing we should do is to separately select tests for each modified branch. All the modified branches for which no test case has been selected are uncovered. These uncovered branches also contain some sort of errors which should be reported besides the failed and crashed test executions.

Why is CC useful, if coverage analysis can be done at any time, and thus we could know this uncovered code part before modifications? To reach 100% code coverage is very difficult or maybe impossible due to infeasible program paths. We should prioritize the uncovered code elements, and create tests for only those which are necessary based on their importance. However, a code part is more error-prone after a code change then before. The more time has passed since the last modification of that part, the higher the probability to introduce some new bugs. By applying continuous coverage analysis, the developer immediately knows that he or she should test the uncovered change.

To cover a branch manually, CC tools may display the existing tests that are „close" to that code part. Close means that only one or at most two predicate outcomes should be modified. Starting from these test cases the manual test generation will be easier, but in some cases it remains still difficult. To address this problem, my imagination of the future involves automated test case construction.

Automated test generation

CC tools produce the branch to be covered. In several cases, however, it is difficult to find test data for which a given code part is executed. Fortunately, tests can be generated - at least in theory -, I personally worked in this field [10]. Even though the generation of all the test cases is very difficult and impractical, it is worth generating some missing test cases automatically.

We can also start from test cases close to the branch to be covered. Thus we have some „almost good" inputs which should be modified to reach our goal. An iterative process selects always the input which is better with respect to our goal. When the code part is covered, the related test is executed and we obtain the output. Note that we rely on the starting test case and we consider the same output variables. Now we have both the input and the output and thus we can generate the missing test.

The test should be executed again and checked by the developer, since this test is automated, but no automation can predict the correct output. The aim of automation is just to cover a given branch, thus the validation of the result is obligatory. If the test actually failed, the correct output should be set by the modification of the test case, and the code should be fixed. Otherwise there's nothing to do and we have improved both the coverage and the efficiency of our test data.

Integrated Development and Test Environment (IDTE) to support agile testing

In the future, traditional IDEs will be expanded to involve testing as well. This extended environment, IDTE should involve continuous testing, continuous coverage and automated test generation tools as well. In addition, ATDD and BDD tools will be much more complex and user friendly in future.

Executable specifications can be written in HTML, wiki or plain text. Our Concordion example is HTML, where the developer or the tester should instrument the specification manually. The instrumented text fragment related to the happy path scenario is as follows.

```
<h3>Scenario happy path</h3>
<p>
        <span concordion:execute="init()"/>
        Given a user with login name „<b concordion:set="#name">Smiths</b>"
        and a password „<b concordion:set="#passw">Shtims2011</b>".
        <span concordion:execute="createAcc(#name, #passw)"/>

        <br/>
        When user is logging in as „<b concordion:set="#name2">Smiths</b>"
        and enter password „<b concordion:set="#passw2">Shtims2011</b>",
        <span concordion:execute="#msg=logsIn(#name2,#passw2)"/>

        <br/>
        Then the message „<b concordion:assertEquals="#msg">Signed in</b>" appears,
        <br/>
        And the user is logged in.
        <span concordion:assertTrue="isLoggedIn(#name2,#passw2)"/>
</p>
```

Though other tools may require simpler instrumentation, the problem is that neither testers nor developers like manual instrumentation, which is also error-prone. A user friendly IDTE will support instrumentation enabling the tester or developer to write simple text as can be seen in Figure 1.
Let us consider how an IDTE can help us.

Since the code beginning with *Given* contains the initial state, at this point available methods can be called. Since the IDTE knows all the scenarios, when entering *Given*, the available scenarios are displayed and the required one can be selected. For example, if we would like to use the login happy path scenario later, then the *When* part (neglecting the keyword) will be displayed:

user is logging in as „**Smiths**" and enter password „**Shtims2010**"

Selecting this we obtain the following fragment:

Given user is logging in as „**Smiths**" and enter password „**Shtims2010**"

Our intention was to log in an existing user. Of course, we can change the values and the text, since after the selection our (unfortunately non-existent) IDTE knows the selected scenario.

The When part is even simpler: since this contains the new action (method), we can just write any text to describe this action, the IDTE will generate the method call with the necessary values:

```
concordion:execute="#msg=logsIn(#name2,#passw2)"
```

The only thing to do is to obtain the input values. Each number is considered to be a parameter (value), while strings and characters are marked with „" as usual. Boolean values are „yes" and „no", while the string yes is expressed as „"yes"". We always assume one method for one action, therefore our „main" method `logsIn` can call others.

Finally, considering the *Then* part, we just write the necessary text to express validation with the expected results, and the IDTE will generate the assert instrumentation. If there are more outputs, we assume an object return value with more instance variables. When this is not the case, as in our example, the developer can modify the instrumented text. However, I note that this executable specification is the *starting point* and all other code can be developed accordingly.

There are several ways to improve an IDTE. For example, if no values are specified a Boolean value *true*, while if the text contains „not", a Boolean value *false* is assumed.

Fixture code generation

In current agile testing when the executable specification has been ready, the next step is to develop the test code called *fixture code*. This is the connection between the executable specification and the code to be implemented. A part of the related fixture is as follows.

```
package com.examples.specs.login;

import org.concordion.integration.junit4.ConcordionRunner;
@RunWith(ConcordionRunner.class)
public class LoginTest {

        private Login login;

        public String createAcc(String name, String passw) {
                String msg = AccountMaker.createAccount(name, passw);
                return msg;
        }

        public String logsIn(String name, String passw) {
                String msg = login.login(name, passw);
                return msg;
        }
```

Some tools generate a template; however, the code should be written by the developers. Though it's not an easy task, it is likely that a large part of the test code can be generated. On the one hand we start from the executable specification, but on the other hand we have freedom since there is no code yet. After the generation, the fixture code should be completed. When the test code is ready, the template code to implement is generated. Only class and method names are generated with parameters. The purpose of this generation is to become the specification really executable. Thus IDTE runs the executable specification which should fail, since the real code is missing behind the tests. From now on, the testing part is ready and coding can start.

Conclusion

In this article my purpose was to describe my vision about the future of agile testing emphasizing methods I consider the most relevant in this field. I believe that agile testing will be much more automated, and supported by an Integrated Development and Test Environment. In this case the boring part of the work will be done by the tool and testers or the developers will do only the interesting part of the process, i.e. developing the executable specification and the target code. Therefore, the specification will be supported in the future in a similar way as coding nowadays. Significantly improving the continuous integration, code will be tested after each modification with save. Uncovered changes are detected just in time, and corresponding test cases are generated. If all these will be fulfilled, agile development will be speeding up, and the quality of the code will increase significantly.

References

[1] http://docs.behat.org/guides/1.gherkin.html

[2] http://concordion.org/

[3] Gojko Adzic: Specification by Example. Manning Publications Co. 2011. ISBN 978-1617290084

[4] K. Beck Test-Driven Development: By Example. Addison-Wesley. 2002. ISBN 978-0321146533

[5] http://dannorth.net/introducing-bdd/

[6] Nachiappan Nagappan & E. Michael Maximilien & Thirumalesh Bhat & Laurie Williams: Realizing quality improvement through test driven development: results and experiences of four industrial teams. In: Empirical Software Engineering Volume 13, Number 3, 2008. pp. 289-302.

[7] Saff, D. and M. D. Ernst, Reducing wasted development time via continuous testing, in: Fourteenth International Symposium on Software Reliability Engineering, Denver, CO, 2003, pp. 281–292.

[8] A. Bertolino. Software testing research: Achievements, challenges, dreams. In L. Briand and A. Wolf, editors, Future of Software Engineering 2007, Los Alamitos, California, USA, 2007. IEEE Computer Society Press.

[9] L. Crispin and J. Gregory: Agile testing, Addison-Wesley. 2009 ISBN 978-0321534660

[10] Á. Hajnal and I. Forgács: An applicable test data generation algorithm for domain errors. In Proc. of the ISSTA'98, ACM Software Engineering Notes 23(2) 63-72, March, 1998.

> biography

Dr. István Forgács
received his PhD in Computer Science in 1993. His research interests include software testing, data flow analysis, slicing and program maintenance. István Forgács has published articles in leading journals and has spoken at conferences. He was program committee member for leading software engineering conferences, such as ISSTA, ESEC-FSE, etc.

After leaving the academy and research, he participated as the inventor and project leader of Y2K.O., the only product which is able to automatically test programs after the Y2K remediation process.

As one of the managing directors of 4D Soft Ltd. he led several significant EU and Hungarian projects, for example:

- *Leader of the consortium of JARTA and JARTA+ projects (static regression tester and analyzer for Java)*
- *Poject leader in JATEST, development of a JAVA continuous testing tool selecting the test cases to be re-executed*
- *Project leader in Hungary of EU IST 6th Framework, ETICS project (continuous integration tool for large distributed systems)*
- *Leader in JIDEBUG project aiming at developing a debugger based on dynamic influences.*

After 25+ years in software testing with both academic and industrial experience, his current focus is on agile projects and leading tool developments.

email: forgacs@4dsoft.hu

Masthead

EDITOR
Díaz & Hilterscheid
Unternehmensberatung GmbH
Kurfürstendamm 179
10707 Berlin, Germany

Phone: +49 (0)30 74 76 28-0 Fax: +49 (0)30 74 76 28-99 E-Mail: info@diazhilterscheid.de

Díaz & Hilterscheid is a member of "Verband der Zeitschriftenverleger Berlin-Brandenburg e.V."

EDITORIAL
José Díaz

LAYOUT & DESIGN
Katrin Schülke

WEBSITE
www.testingexperience.com

ARTICLES & AUTHORS
editorial@testingexperience.com

350.000 readers

ADVERTISEMENTS
sales@testingexperience.com

SUBSCRIBE
www.testingexperience.com/subscribe.php

PRICE
online version: free of charge -> www.testingexperience.com
print version: 8,00 € (plus shipping) -> www.testingexperience-shop.com

ISSN 1866-5705

Picture Credits

Index of Advertisers

Subscribe for the printed issue!

Please fax this form to +49 (0)30 74 76 28 99, send an e-mail to info@testingexperience.com or subscribe at www.testingexperience-shop.com:

Billing Adress

Company: _____

VAT ID: _____

First Name: _____

Last Name: _____

Street: _____

Post Code: _____

City, State: _____

Country: _____

Phone/Fax: _____

E-mail: _____

Delivery Address (if differs from the one above)

Company: _____

First Name: _____

Last Name: _____

Street: _____

Post Code: _____

City, State: _____

Country: _____

Phone/Fax: _____

E-mail: _____

Remarks: _____

1 year subscription

☐ **32,– €**
(plus VAT & shipping)

2 years subscription

☐ **60,– €**
(plus VAT & shipping)

Date

Signature, Company Stamp

Training with a View

Díaz Hilterscheid

Dates *	Course	Language	Place	Price
22.02.12–22.02.12	Anforderungsmanagement	German	Berlin	€ 499[1]
13.02.12–17.02.12	CAT – Certified Agile Tester (German Exam)	English	München	€ 2,100
05.12.11–09.12.11	CAT – Certified Agile Tester (German Exam)	English	Berlin	€ 2,100
30.01.12–03.02.12	CAT – Certified Agile Tester	English	San José, USA	$ 3,350
19.01.12–20.01.12	HP Quality Center	German	Berlin	€ 998
14.12.11–15.12.11	HP QuickTest Professional	German	Berlin	€ 998
20.02.12–21.02.12	HP QuickTest Professional	German	Berlin	€ 998
23.01.12–25.01.12	IREB® Certified Professional for Requirements Engineering – Foundation Level	German	Mödling, Austria	€ 1,400
14.02.12–16.02.12	IREB® Certified Professional for Requirements Engineering – Foundation Level	English	Stockholm	€ 1,600
13.02.12–17.02.12	ISTQB® Certified Tester Advanced Level – Technical Test Analyst	German	Berlin	€ 2,100
30.01.12–03.02.12	ISTQB® Certified Tester Advanced Level – Test Analyst	German	Berlin	€ 2,100
09.01.12–13.01.12	ISTQB® Certified Tester Advanced Level – Test Manager	German	Berlin	€ 2,100
27.02.12–02.03.12	ISTQB® Certified Tester Advanced Level – Test Manager	German	Suttgart	€ 2,100
27.02.12–02.03.12	ISTQB® Certified Tester Advanced Level – Test Manager	German	Karlsruhe	€ 2,100
05.12.11–08.12.11	ISTQB® Certified Tester Foundation Level	German	Stuttgart	€ 1,800
16.01.12–19.01.12	ISTQB® Certified Tester Foundation Level	German	München	€ 1,800
16.01.12–19.01.12	ISTQB® Certified Tester Foundation Level	German	Karlsruhe	€ 1,800
12.12.11–14.12.11	ISTQB® Certified Tester Foundation Level – Kompaktkurs	German	Frankfurt	€ 1,495
03.01.12–05.01.12	ISTQB® Certified Tester Foundation Level – Kompaktkurs	German	Berlin	€ 1,495
31.01.12–02.02.12	ISTQB® Certified Tester Foundation Level – Kompaktkurs	German	Frankfurt	€ 1,495
06.02.12–08.02.12	ISTQB® Certified Tester Foundation Level – Kompaktkurs	German	Mödling, Austria	€ 1,495
20.02.12–22.02.12	ISTQB® Certified Tester Foundation Level – Kompaktkurs	German	Munich	€ 1,495
20.02.12–22.02.12	ISTQB® Certified Tester Foundation Level – Kompaktkurs	German	Nürnberg	€ 1,495

*** subject to modifications**
all prices plus VAT

IREB International Requirements Engineering Board

CAT Certified Agile Tester®

ISTQB® Accredited Training Provider

Kurfürstendamm, Berlin © Katrin Schülke

more dates and onsite training worldwide in German, English, Spanish, French on our website:
http://training.diazhilterscheid.com