7

September, 2009

testing experience

The Magazine for Professional Testers

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

Agile Testing

© iStockphoto/Mlenny

© iStockphoto/Yuri_Arcurs

ONLINE TRAINING

English & German (Foundation)

ISTQB® Certified Tester Foundation Level

599,- €
incl. exam & plus VAT

Advanced Level now available - special introductory offers
(until 30th September)

**ISTQB® Certified Tester
Advanced Level Test Manager**

749,- €
incl. exam & plus VAT

**ISTQB® Certified Tester
Special Combined Foundation
plus Advanced Test Manager**

999,- €
incl. exams & plus VAT

Our company saves up to

60%

of training costs by online training.

**The obtained knowledge and the savings ensure
the competitiveness of our company.**

www.testingexperience.learntesting.com

ACCREDITED ISTQB TRAINING PROVIDER

ISTQB® International Software Testing Qualifications Board

Díaz Hilterscheid

te testing experience

Dear readers,

Alessandro Collino is the one who inspired me to devoting an issue to Agile Testing.

I have been following the Agile scene for a long time, especially in USA, but was not really connected to it directly, although some employees of mine got the Scrum Master Certification some years ago and work in Agile projects.

The magazine and the great articles in it has given me the opportunity to establish the missing strong link to the Agile community and read about detailed experiences and opinions.

I don't know if all pigs are equal or not, but have a look at Stuart Reid's article. I like it very much.

I also like the statement of Lior Friedman: "The only way to go fast is working in high quality".

I think – and I am probably not alone on that – that knowledge is the key to success. If you or your team don't have the knowledge, then it is difficult to get the high quality.

On one hand you must have the process knowledge on Agile, but on the other hand you have to know how to deal with testing. It doesn't matter whether you are using exploratory techniques, systematic techniques or if you automate your tests or not. Your team needs the testing skills, if you want to move forward fast.

Based on this topic we decided to make a call for papers for the Agile Testing Days in Berlin. The response was great. I think that one of the key points was/is that very good professionals like Lisa Crispin, Elisabeth Hendrickson, Tom Gilb, Stuart Reid, Isabel Evans, Mary and Tom Poppendieck and, last but not least, Alessandro Collino were involved from the beginning. Their tutorials and the Conference will take place from October 12th to 14th. For your registration go to: www.agiletestingdays.com

According to our vision of the testing community, we think that the support of conferences, events etc. from other vendors is good for the testing world, especially if they give the readers a good discount. Check our website. You will always find some special rates for "testing experience" readers. Please send me your comments on the visited events.

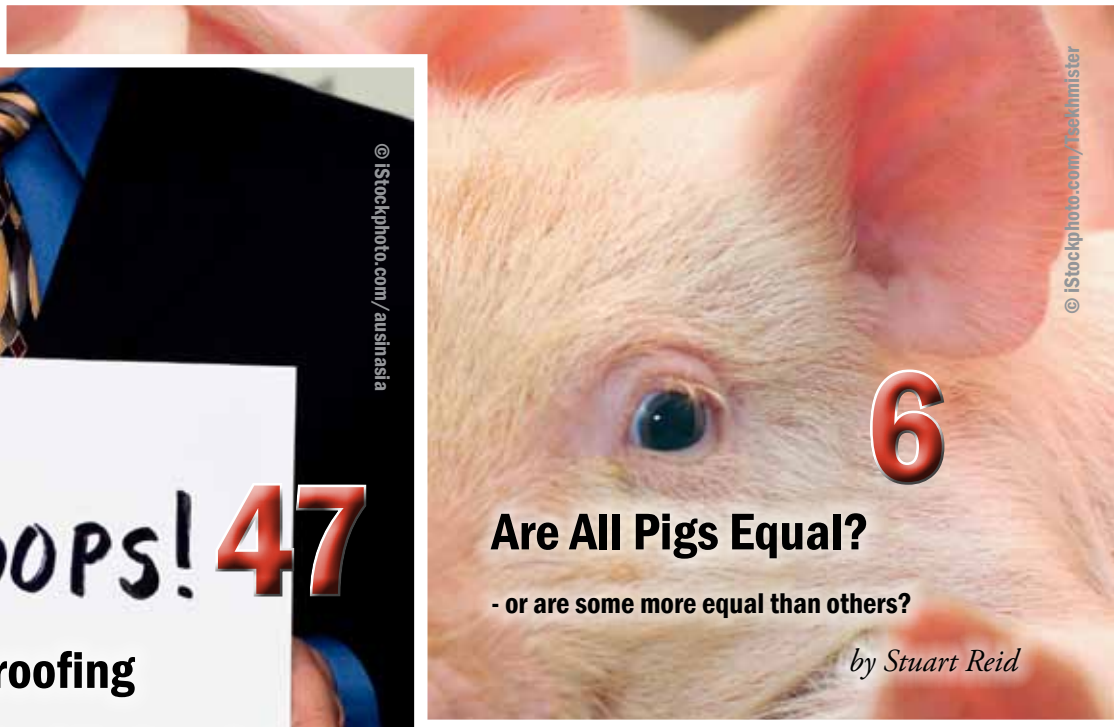
Finally, let me just tell you that I was in Gran Canaria with my son spending some days at the Las Canteras beach snorkeling. It was amazing to feed fish while snorkeling with thousands of them dancing around you! The salty water of the Atlantic ocean is something that should not be missed. It is really a family beach. I will be back there in October before the conference and will enjoy it again. Come and see me for a beer or a coffee.

Enjoy the magazine and hope you had a good summer.

Kind regards

José Manuel Díaz Delgado

Contents




© iStockphoto.com/Tsekhmister

Are All Pigs Equal?

- or are some more equal than others?

by Stuart Reid

"ALL ANIMALS ARE EQUAL BUT SOME ANIMALS ARE MORE EQUAL THAN OTHERS."

George Orwell, Animal Farm, 1945.

Imagine you are offered the opportunity to take up one of two jobs that are similar in all respects, except one is working on a traditional project and the other is working on an agile project. Which one would you take?

One thing is practically certain - most developers would jump at the opportunity to work in an agile environment. But would (or should) a tester?

"How agile is this project?"

As a tester one of the first questions you would need to ask is "how agile is this project?" This begs a further question: "Are there levels of project agility?" An agile purist would point to the agile manifesto (see below) and the associated principles and argue that a true agile project would be aligned with all of these. The reality is that there are very few truly agile projects out there; in practice the label of 'agile' seems to be accepted as long as the project develops software incrementally with the delivery of each increment typically taking no longer than 4 weeks. Alignment with the other principles varies dramatically. So the question of 'how agile?' is legitimate, but how does level of project agility affect testers? To answer that question you need to understand that a fundamental difference between working on an agile and a traditional project is that an agile team is empowered to make decisions and everyone is jointly responsible for the output of the team.

Agile team bonding

Everyone on an agile team works together towards a single goal and the best agile teams are those where the management have been able to provide an environment that nurtures the feelings of empowerment, togetherness, joint responsibility and trust within the team. In my experience of talking to team members on successful agile project teams, it is these characteristics that they generally find most attractive. Ensuring this healthy team environment is rarely cited as the main objective of organizations adopting an agile approach, but it is often seen as a key attribute associated with successful agile projects.

Testing outside the agile team

An integral part of building a cohesive agile team is the shared responsibility for the team's output. The delivery of useful, operational software on a regular and frequent basis is a goal of a pure agile project. If the team's output is going to be a fully-tested, usable piece of functioning software (as declared in the 'Principles behind the Agile Manifesto' – see http://agilemanifesto.org/principles.html) then testing must be an integral part of the team that produces it – but many supposedly agile projects do not deliver usable software at the end of each cycle. Instead, they deliver software that still needs to be tested in a realistic environment, has not been tested against non-functional requirements such as performance, and also still needs to be user acceptance tested.

Thus on many 'agile' projects we find that the necessary specialist testing is not performed within the agile development team, but instead done as a separate activity some time after the agile development team delivers their output. So the main reason a tester needs to ask the question "how agile is this project?" is to determine whether they are going to be embraced into the togetherness of the agile team. The alternative is they are given an outsider's role of checking the agile development team's outputs and feeding incident reports back to them while the development team are only really interested in concentrating on creating their next increment – much the same as with traditional development approaches. In this situation the tester would not even be considered a pig, let alone an equal pig, but would more likely be considered a chicken (see right).

Manifesto for Agile Software Development

We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:

- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

That is, while there is value in the items on the right, we value the items on the left more.

Kent Beck	*James Grenning*	*Robert C. Martin*
Mike Beedle	*Jim Highsmith*	*Steve Mellor*
Arie van Bennekum	*Andrew Hunt*	*Ken Schwaber*
Alistair Cockburn	*Ron Jeffries*	*Jeff Sutherland*
Ward Cunningham	*Jon Kern Dave Thomas*	
Martin Fowler	*Brian Marick*	

© 2001, the above authors this declaration may be freely copied in any form, but only in its entirety through this notice.

Multi-functional agile teams

Suppose it's good news and the project is more agile than most and delivering usable software at the end of each sprint (the majority of projects that label themselves as 'agile' do not deliver usable software at the end of each sprint). In this situation the testers will necessarily be an integral part of the agile team and thus pigs. So far, so good – you'll presumably get all the benefits of being fully bonded into a successful team. You now need to establish what your expected role will be. If we return to the purist view we may well find that there is an expectation that all agile team members are able to perform any activity needed in the team – the so-called 'multi-functional' team. So, one day you may be programming, another day testing, and a third helping subject matter experts write user stories. This is one of the least likely 'pure agile' approaches you may encounter, but worth checking on, in case this happens to be one of these rare projects. For many testers this is going to be a deal-breaker – because "if they could design and program (and get paid more for doing so) they probably wouldn't be testers, would they?"

Despite the ideal of a multi-functional team and the corresponding benefits it creates with planning, scheduling and reviewing, practically no agile teams achieve this 'nirvana'. It is worth noting, however, that those teams that aspire to this are often perceived to be good places to work in as they expect and support team members in their continued professional development to become more widely skilled. Imagine the pleasure of working in a team where the developers are trying to understand how they can improve their testing practices, and where testers are treated as equals as they have demonstrated their ability to add value in design and code reviews using the development skills they have acquired.

An agile development and test process?

So, you decide to take on the nominal role of test analyst within an agile team. What can you expect to be your responsibilities? Let's consider what testing typically gets done in an agile sprint. Testing needs to be aligned with development (even in agile) so one place to start would be considering what development activities take place. In fact, it would be useful if we could define a generic tailorable process for agile projects and then we could see both the development and test activities and how they inter-relate (see figure below).

Those of you who read the agile manifesto earlier (or already know it) may well be wondering how I can even consider defining a process in an article on agile. "*Individuals and interactions over processes and tools*" is right at the start of the manifesto. It states at the end of the agile manifesto that the stuff on the right is considered of lower value, but I find that everyone knowing what process they are following is not just useful but absolutely crucial. For me the major difference with the process in agile projects is not that it is of less value, but rather that it is flexible and not a fixed process that rarely changes as in traditional projects. In an agile project the team should be empowered to evolve and (hopefully) improve their process, perhaps as often as after each increment.

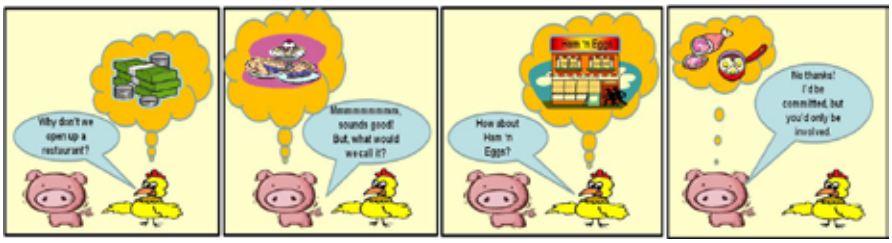
Testing roles in agile teams

In the provided model of a generic agile development and test process, testing is explicitly shown in a number of stages. Of course, not all stages will be required in all sprints and not all the testing shown in this process would be the responsibility of the tester. For instance, the testing during the 'develop stories' stage would normally be the responsibility of the developers, although the tester would be expected to provide advice in this area, as necessary. Similarly the acceptance test stage would ideally be executed by the end users, but the testers would normally play an advisory role to these users (e.g. how to write good acceptance tests) and take responsibility for setting up this stage of testing.

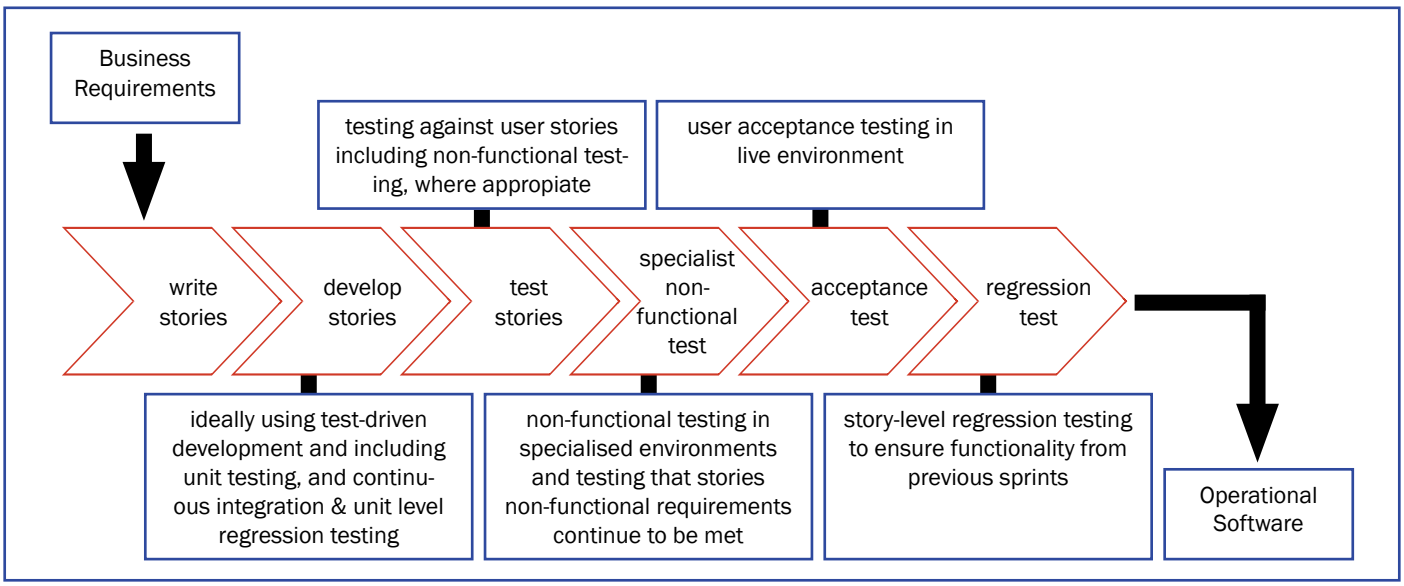
Testers would normally take prime responsibility for the 'story testing' (testing against user stories) and the story-level regression testing, which will involve running a regression test suite that is built up from the story tests of previous sprints. While much of the non-functional testing would be performed as

Pigs & Chickens

The various stakeholders working on and with agile projects are often referred to as pigs and chickens; you are either one or the other, and note that neither term is supposed to be offensive. This classification is based on the joke in the cartoon below. Those stakeholders who spend all their time on the project are known as pigs (e.g. the sprint team members), while the remaining stakeholders (e.g. domain experts, who are occasionally approached for advice) are known as chickens. If you hold a daily scrum meeting then the pigs are expected to have their say, while the chickens are expected to listen.



The role of the product owner or customer representative can be that of either a pig or chicken dependent on whether they devote all or just some of their time to the project. Similarly, testers can be either pigs or chickens dependent on whether they are an integral part of the sprint team or used as a separate testing service after the sprint has finished.



Agile TESTING DAYS

Berlin, Germany

The Agile Testing Days are the European conference for the worldwide professionals involved in the agile world.

We decided to choose Berlin as one of the best connected capitals in Europe and also for its very good price/service ratio for hotels and flights. Berlin offers also a lot of fabulous places to visit in your spare time.

Please have a look at the program (www.agiletestingdays.com) and enjoy the conference!

October 12
Tutorials

"Using the Agile Testing Quadrants to Cover Your Testing Needs"
by Lisa Crispin

"Managing Testing in Agile Projects"
by Isabel Evans and Stuart Reid

"State of the art and future of Agile Testing"
by Alessandro Collino

"Real QA - Real Software and System Quality Assurance"
by Tom Gilb

"Acceptance Test Driven Development (ATDD) in Practice"
by Elisabeth Hendrickson

"Designing a Lean Software Development Process"
by Mary and Tom Poppendieck

October 13
Conference

Key Note - "Are Agile Testers Different?"
Lisa Crispin

"Agile Testing: A Report from the Frontline"
Joke Hettema, Ralph van Roosmalen

"Open Source Agile Testing"
Péter Vajda

The Agility GPS
Ulrich Freyer-Hirtz

BDD approaches for Web Development
Thomas Lundström

Introduction to Robot Framework"
Pekka Klärck

"How to develop a common sense of 'DONE'?"
Alexander Schwartz

"Key Note"
Elisabeth Hendrickson (TBD)

"Agile Quality Management –Axiom or Oxymoron?"
David Evans

"HtmlUnit: An Efficient Approach to Testing Web Applications"
Marc Guillemot

"Quality and Short Release Cycle"
Lior Friedman

"Identifying the value of Performance testing in an agile world!"
Mieke Gevers

"'Testify' – One-button Test-Driven Development tooling & setup"
Mike Scott

"Test Driven TDD - TDD quality improvement based on test design technique and other testing theory"
Wonil Kwon, Hyungil Cho

Key Note - "Agile Inspections"
Tom Gilb (TBD)

October 14
Conference

Key Note - "The One Thing You Need to Know ... About Software Development"
Mary and Tom Poppendieck (TBD)

"Improved Agile Testing using TPI"
Cecile Davis

"Configure a build server in 60minutes"
Emanuele DelBono, Alessandro Melchiori

"Learning is key to Agile success: Building a learning culture on your Agile team"
Declan Whelan

"The Missing Link for ATDD & Example-driven Development"
Gojko Adzic

"Promoting the use of a quality standard"
Eric Jimmink

"Test Driven Development of Embedded Software at MiPlaza"
Marcin Czenko, Wim van de Goor, Martijn Gelens

Key Note - "Investing in individuals and interactions"
Stuart Reid (TBD)

"Bridging the gap between agile and traditional testing"
Anko Tijman

"Automated Integration Testingin Agile Environments"
Slobodanka Sersik, Dr. Gerald Schröder

"Agile practices in a traditional environment"
Markus Gärtner

Clossing Session José Díaz & Alessandro Collino - Podium Diskussion

a Díaz & Hilterscheid Conference

Please fax this form to +49 (0)30 74 76 28 99
or go to http://www.agiletestingdays.com/onlinereg.html.

Agile TESTING DAYS

Participant

Company: _____

First Name: _____

Last Name: _____

Street: _____

Post Code: _____

City, State: _____

Country: _____

Phone/Fax: _____

E-mail: _____

Billing Address (if differs from the one above)

Company: _____

First Name: _____

Last Name: _____

Street: _____

Post Code: _____

City, State: _____

Country: _____

Phone/Fax: _____

E-mail: _____

Remarks/Code: _____

Tutorial

- ☐ Using the Agile Testing Quadrants to Cover Your Testing Needs, by Lisa Crispin
- ☐ Managing Testing in Agile Projects, by Isabel Evans and Stuart Reid
- ☐ State of the art and future of Agile Testing, by Alessandro Collino
- ☐ Real QA - Real Software and System Quality Assurance, by Tom Gilb
- ☐ Acceptance Test Driven Development (ATDD) in Practice, by Elisabeth Hendrickson
- ☐ Designing a Lean Software Development Process, by Mary and Tom Poppendieck

700,- € (plus VAT)

Conference

- ☐ 1 day (first day) 550 EUR
- ☐ 2 days 850 EUR
- ☐ 1 day (second day) 550 EUR

850,- € (plus VAT)

Included in the package: The participation on the exhibition, at the social event and the catering in course of the event.

Notice of Cancellation

No fee is charged for cancellation up to 60 days prior to the beginning of the event. Up to 15 days prior to the event a payment of 50% of the course fee becomes due and after this a payment of 100% of the course fee becomes due. An alternative participant can be designated at any time and at no extra cost.

Settlement Date

Payment becomes due no later than the beginning of the event.

Liability

Except in the event of premeditation or gross negligence, the course holders and Díaz & Hilterscheid GmbH reject any liability either for themselves or for those they employ. This also particularly includes any damage which may occur as a result of computer viruses.

Applicable Law and Place of Jurisdiction

Berlin is considered to be the place of jurisdiction for exercising German law in all disputes arising from enrolling for or participating in events by Díaz & Hilterscheid GmbH.

Date Signature, Company Stamp

part of the story testing, some may also be performed during unit test (e.g. memory management). However, occasionally specialist test skills or environmental constraints may mean that a separate stage is required.

It should also be noted that the order of processes shown in the development and test process model need not be followed exactly, as some projects, for instance, will perform the story-level regression testing more than once in a sprint, and may end the sprint with acceptance testing.

Different testing skills?

In many respects the testing processes followed in an agile sprint are much the same as those performed during a traditional project, as the testing still has to follow a fundamental test process. Differences typically arise in two areas: automation and the test basis (i.e. how the software under test is specified).

User stories

Unsurprisingly, agile projects prefer a lean approach to specification, which typically means that specifications (user stories) are far shorter than in (well-run) traditional projects (but remember bigger is not always better!). This economy of documentation is partially offset by the fact that the author of the user stories is normally close at hand and can be questioned directly when issues arise. In many agile projects testers are closely involved in the story writing, teaming up with business analysts to ensure that the stories are complete and testable – and ideally helping to define acceptance criteria up-front.

Isn't agile testing all exploratory?

A natural and common response by testers to poor specifications is to go for an exploratory testing approach, but this scenario should not occur on agile projects. Not because exploratory testing is inappropriate, but because poor specifications should be rare. This situation does not, however, just happen because we have labelled the project as 'agile' – the agile team need to ensure it happens by adopting a process that supports the creation of user stories that include the minimum amount of information needed to both develop and test the requirement described by the story. One way of doing this is for a tester to be directly involved when the stories are created, so guaranteeing that the stories take into account and support the tester's perspective. Another way is for the story template to require authors to explicitly include acceptance criteria with the story and for the team to never accept stories into a sprint that are not complete in this respect.

Unless the agile project is dysfunctional and adequate user stories are unavailable then the testing in an agile project is not just exploratory (see 'agile testing' below). You need to have the full range of test techniques available to you, so that you can select the right one for the situation. Agile projects produce software for a wide variety of applications, some of which need to meet regulatory requirements and some of which may be safety-critical, meaning we can't just rely on a single approach with no repeatability. Exploratory testing may often be an appropriate choice, but rarely will you use exploratory alone; I would normally expect it to be used to complement more systematic techniques. And, of course, where we create automated tests (discussed on next page) these tests are necessarily all scripted (and so not exploratory) encouraging us to use an exploratory approach to provide balance.

Agile Testing

The misunderstanding that all testing on an agile project is exploratory is perpetuated by misuse of the term 'agile testing' as a synonym for 'exploratory testing'.

AGILE TESTING ≠ EXPLORATORY TESTING

The term 'exploratory testing' was coined in 1983 by Cem Caner, and the following description is provided by James Bach, a leading advocate:
Exploratory testing is simultaneous learning, test design, and test execution.
In other words, exploratory testing is any testing where the tester actively controls the design of the tests as those tests are performed and uses information gained while testing to design new and better tests.
If, at one extreme, fully scripted testing is performed where all tests are designed up front and test execution is simply running these pre-designed tests then exploratory testing provides the opposing view of this. In practice exploratory testing does not have to be completely spontaneous and 'partially planned' approaches such as session-based (exploratory) testing have been found to be very successful.
Exploratory testing as an approach can be considered to align closely with the spirit of the agile manifesto, but care should be taken to not equate exploratory testing with the testing performed on agile projects. It is widely accepted that the most effective test strategies (and this applies to both agile and non-agile projects) include both systematic scripted techniques and exploratory testing. In fact, given the reliance of agile projects on test automation (which inevitably uses scripted tests), it becomes obvious that agile projects cannot use a test strategy based solely on exploratory testing.
Thus, given that most people associate the term 'agile testing' with the testing performed on agile projects, it becomes clear that agile testing cannot be considered another term for exploratory testing. Conversely, exploratory testing can be considered to be an agile approach to testing.

Test Automation Skills

Although automation is common in traditional projects, it is an absolute necessity in agile projects (despite it being on the 'of less value' right-hand side of the manifesto). Test-driven development, automated builds and continuous regression testing all rely upon automation and won't work without it, while story-level regression testing and acceptance testing are also automated on many agile projects.

Do not worry, however, if you are not a tools expert. On many agile projects the test analyst does not take responsibility for supporting test automation as this is often considered to be the responsibility of either a specialist tools developer or a 'part-time' job for one of the developers. So not being able to program and not being a test automation specialist does not disqualify testers from working on most agile projects.

Lean development and testing

It appears as though developers get a definite benefit from the use of the lean approach advocated on agile projects as their requirement to document is substantially reduced – and who likes creating documentation? You might be asking yourself if there's a similar benefit to be gained by the testers – and there is.

A typical attribute of a successful agile project is a co-located team where each team member can speak to another without having to move from the team area (and ideally their desk). This introduces the possibility that when testers identify a potential issue with the software they simply go to talk to the relevant developer about the issue. This works best when testing is performed as soon after the developer releases the software to test as possible. If the issue can be immediately resolved by the developer then it can be argued that there is then no need for the issue to be recorded in the project's incident management system. Of course, where the issue cannot be resolved within the

For us, Agile is more than a buzzword ...

SELA – Leading in offering blended management solutions, based on traditional and Agile methodologies

Becoming Agile is no longer optional for organizations wishing to stay competitive in today business. SELA helps organizations to perform such a transition as smoothly as possible. Our combination of unique courses, consulting services and practical expertise helps our customers to speed up the transition and maximize the benefits of becoming more Agile. We will help you choosing the ideal combination of Agile and traditional methods that fits most your business needs

Available Courses:

- Agile Testing
- Practical Scrum
- Scrum Master Certification Course
- Scrum Product Owner Certification Course
- Test Driven Development for C++ Developers
- Test Driven Development for Java Developers
- Test Driven Development with .NET
- Test Driven Development with VSTS

Solutions are available around the world. Local solutions are also available, in:

Argentina | Canada | Germany | India | Israel | USA | Singapore

solutions@sela.co.il
www.sela.co.il/en

SELA TECHNOLOGY CENTER

Microsoft GOLD CERTIFIED Partner
Learning Solutions
Networking Infrastructure Solutions
Information Worker Solutions
Custom Development Solutions

current sprint then it needs to be documented. The counter argument for recording every issue found by testing often cites the situation where a developer may just nod to the tester but then subsequently ignore or forget the issue. This argument, however, is flawed as the software will not get signed off as tested until it passes the tests, which it currently has not and will not do until the issue is resolved and the software retested.

No incident reports?

A traditionalist may well find this a difficult concept as previously they will have been forced to diligently document all issues and may well argue that without metrics no improvement can take place. However, in an agile project a retrospective is held at the end of each sprint to agree how improvements can be made and most suggestions will be supported by argument based on what happened in the last few weeks rather than an analysis of incident reports collected over a prolonged period. Perhaps the most persuasive argument I have heard for recording all incidents was made by two testers on a successful agile project whose boss was a traditionalist. His major input to their annual appraisal was the number of issues they had raised during the last year! In contrast, I find the most persuasive argument for not collecting fault metrics is to ask those who advocate their collection when they last found time to analyse them and act on the results.

The tester benefits on two fronts from not having to document all issues they discover. First, they don't lose the time that is spent on writing up each bug report. Second, there is less chance that their bug reports will cause an 'over the wall' division in the team between developers, who see themselves as being positive and moving towards the goal of delivering usable software, and testers, who are seen as simply trying to slow this process down. On an agile team all team members are supposed to be working together towards a single goal and, if done with sensitivity, verbally communicated issues are easier to accept than those received from an anonymous incident management system.

So are all pigs equal?

George Orwell's anti-Stalinist novel, Animal Farm, begins with the animals declaring their equality, but later the pigs form an elite and move to "all animals are equal but some animals are more equal than others". In an agile project we have already seen that the sprint team (all who are fully involved) are referred to as pigs, but are all members of the sprint team considered equal? It certainly appears as though the developers get major benefits from working on agile projects and most of the drive towards agile comes from developers. But, do the testers in an agile project team also gain from going agile?

The adoption of a lean approach in agile means that the amount of documentation is kept to a minimum, which is perceived as a benefit by all those that have to generate it. A concern of testers is that this is taken too far and inadequate documentation is available to support testing (and future changes), but in a well-run agile project practices should evolve to ensure a happy medium is achieved. Of course, the extension of this lean philosophy to incident management means that testers gain a similar benefit.

Test-driven development (TDD) is an advantage to both the developers and the project as a whole. Once they have grasped how it works nearly all developers embrace it as the only way they want to write software. Lead developers also like it as even their less capable programmers seem to produce reasonable quality code when using it. TDD generally results in higher quality code and creates automated tests as a by-product; these can then be used for automated unit level regression testing at practically no extra effort. This practice also raises developers' awareness of testing, which can only be of benefit to the testers.

As mentioned earlier, the team spirit and trust in a well-run agile team is considered a major benefit by those working in it; this applies equally to the testers as long as they are embedded as part of the sprint team. In this situation, the tester shares with the rest of the team in the responsibility for the team's outputs. In a successful team this means they also share in the reward of knowing they are producing something useful to the users, and, importantly, they get this (hopefully) positive feedback on a frequent and regular basis.

The best deal

In many respects I believe the testers get the best deal of all those working in an agile team, although this is partly based on comparing their agile situation to that on traditional projects. In agile they typically take part in a wider

Tester's Agile Checklist

Answer each question 'yes' or 'no' then check your score using the grid below.		Y/N
1	Would you be offended by being classified as a pig?	
2	Do you consider exploratory testing to be the equivalent of hacking?	
3	Are the testers on the project considered to be part of the sprint team?	
4	Do you find contentment in writing fully documented incident reports?	
5	Is the output of each sprint immediately usable by the users?	
6	Are you happiest when working alone?	
7	Are you too shy to question decisions made by developers and business analysts?	
8	Do you find that regression testing is a waste of time?	
9	Are all projects in this part of the organization agile?	
10	Does the project have both a scrum master and a project manager?	
11	Are the developers using test-driven development?	
12	Do you find yourself panicking when faced with short deadlines?	
13	Are continuous integration and automated regression testing implemented on the project?	
14	Is exploratory testing all you want to do?	
15	Do you consider test automation to be someone else's problem?	
16	Can you see yourself sitting at a terminal and working on code with a developer?	
17	Does the thought of a lack of detailed specifications make you uneasy?	
18	Do you stick with your plans no matter what?	
19	Are you willing to take joint responsibility with the rest of the sprint team for the deliverables?	
20	Are you happiest following a fixed set of procedures rather than looking for a better way?	

Award yourself one mark for each of your answers matching those shown below.

1	N	6		N	11		Y	16		Y
2	N	7		N	12		N	17		N
3	Y	8		N	13		Y	18		N
4	N	9		Y	14		N	19		Y
5	Y	10		N	15		N	20		N

Scores of 15 or more suggest you should accept the challenge of working on an agile project!

variety of tasks, even over the short duration of a typical sprint. They get to interact closely with the developers and BAs and can even get direct access to the users on a regular basis. While working on story writing they can expand their analysis skills, while the emphasis on automation in agile allows them to improve their skills in test tools and scripting if that is an area they are interested in. Many agile teams use a pair-programming approach, which provides the perfect opportunity for testers to pick up and demonstrate programming skills, potentially allowing them to become true multi-functional agile team members.

If you consider there to be a career progression within an agile team then the most obvious move is from being a team member to scrum master. There is no reason that a tester could not make this move as easily as any other agile team member given the necessary experience, and some of the most successful scrum masters I have seen have graduated to that position by way of testing.

Conclusion

So, if you are offered the choice of testing on a traditional or an agile project, which way should you go? That depends. If you've got this far through the article it is safe to assume you are not a software tester who just does the minimum they need to do each day and who has no interest in making their jobs and lives more interesting (if you read articles on software testing that probably puts you in the minority of people in the software testing profession). So, my guess is you'd probably be better off going for the agile option.

But, it is not just a question of whether you are ready to work on agile projects. There is also the question of whether the project is agile *enough*. At the end of George Orwell's Animal Farm the pigs have reduced the commandments to the single "ALL ANIMALS ARE EQUAL BUT SOME ANIMALS ARE MORE EQUAL THAN OTHERS." There are now many projects that label themselves as 'agile', but some agile projects are more agile than others. If I were joining an agile project as a tester I would need to know that the testers are treated as pigs and that all pigs are treated as equals.

If you want further help on making your choice then fill in the completely unscientific Tester's Agile Checklist and see which way it takes you.

Biography

Dr Stuart Reid FBCS CITP
Stuart is Chief Technology Officer at Testing Solutions Group, providing consultancy and training in software testing worldwide. He is convener of the ISO Software Testing Working Group developing the new ISO 29119 standard, and chairs the BCS Specialist Group in Software Testing. Stuart founded the ISTQB and still works in this area. He was awarded the EuroSTAR Testing Excellence Award in 2001 and presents papers and tutorials at conferences worldwide.

MAKE FiT®
iSQI's Education Campaign

iSQI Education Campaign "MAKE FiT" 2009

Education instead of crisis! True to the motto "MAKE FiT" iSQI champions the further education of professionals from the IT industry and thus takes the initiative for a successful 2009.

We grant an education loan without repayment.

For many years now iSQI has stood for high-quality advancement of skilled personnel. The institute is certified according to ISO 9001:2008, its certification processes are audited to be in accordance with ISO 17024. We want to use our long-time experience to help as many employees as possible to get through the crisis unscathed. Hence we will actively participate in the costs of our clients' further education. **Please talk to us.**

Deals

www.makefit.org

Everyone who signs up for a certification exam at the International Software Quality Institute (www.isqi.org) in September 2009 will receive a one day ticket for the CONQUEST Conference on Quality Engineering in Software Technology (www.conquest-conference.org) in September in Nuremberg.

Contact: Malte Ullmann **(malte.ullmann@isqi.org).**
Please send an e-mail with the subject line: **"MF Cert+Conf"**

All-clear for „hackers“

by Julia Hilterscheid

The Federal Constitutional Court of Germany recently had to decide whether simulated hacker attacks are liable to prosecution.

The appellants had raised the question whether the preparation of the spying out and intercepting of data (and as a result § 202 c StGB (German penal code) which makes this a punishable offense) are in conformance with the German Constitution. In addition to other basic rights, the German Constitution guarantees the freedom of professional occupation and the freedom of teaching, which are affected in this context.

One of the appellants is Chief Executive Officer of a company offering services in the fields of information security and communication technologies. In order to test the security of clients' computer systems through simulated hacker attacks, the employees of the company also perform penetration tests. Typically, the specialists attack the client's network with the objective of hacking into the client's data processing system to be tested. These activities are performed exclusively on behalf of and with consent of the system operator, who wants to correct any security vulnerabilities detected through these attacks.

Another appellant is a professor at the University of Applied Sciences Berlin in the faculty of computer sciences and media. In the course of his lectures he first communicates knowledge about the use of so-called security analysis tools. These software programs are designed to detect possible weak points in data processing systems through systematic testing and perform a security analysis of the vulnerabilities of the tested systems. For practical exercises, the programs are handed over to the students on a data carrier. In addition to this, the students can also download the programs from the university professor's homepage. Via this homepage, however, it is also possible for third parties to download the programs, which cannot be controlled by the appellant. The software posted on the net via the homepage is either software that is in any case freely available in the internet, or software programs developed or modified by the appellant himself.

Problematical under criminal law aspects is in this context not only the access to the appellant's internet page through third parties, which cannot be excluded, but also the fact that the programs can generally be used for criminal purposes to hack into other people's computer systems and networks.

Whilst it had already been clarified prior to the decision elaborated in this article that hacking as such is liable to prosecution, it has been possible since the introduction of the so-called "hacker paragraph" two years ago to also punish the preparatory activities for spying out and intercepting of data with up to one year imprisonment or with a fine. This concerns offenders who prepare the spying out and intercepting of data by making passwords or other security codes protecting the data accessible or by making computer programs for criminal purposes accessible to themselves or to third parties.

Ms. Hilterscheid has been solicitor with practising licence since 1997. After stays abroad and studies in law in Berlin, she founded first the solicitor's office Hilterscheid, and one year later, together with her husband, the consultancy Díaz & Hilterscheid Unternehmensberatung GmbH. In addition, Ms. Hilterscheid has been teaching media law and copyright at the Berlin University of Applied Science.
Ms. Hilterscheid has been supporting enterprises for more than 10 years in the discretionary formulation of contracts, general terms and conditions and license agreements, and accompanies projects legally. She ensures for her clients that legal requirements are adhered to in their correspondence, on-line appearance as well as in their advertising and marketing activities.
Ms. Hilterscheid also offers seminars on the subjects of IT-law, trademark law, copyright and labor law, which can take place in-house if so desired.

www.kanzlei-hilterscheid.de

Column: The Court!

Despite the fact that hacking is already liable to prosecution, interested persons can without difficulty find suitable programs on the internet which can be used, among other things, for breaking passwords or the serial numbers of commercial software products. There is also access, for example, to software which can be used for server attacks (denial of service) or to so-called virus construction kits, which can be used to overload servers or to take them out of service.

The mere penetration of a system is usually exempt from punishment. If, however, data is manipulated in this act, or if the penetrator procures specially secured data in the course of the activities, punishment of the actions may be considered. In this context, it is not relevant as a matter of principle, whether the penetrator had no intention of committing a criminal act, as in the case of the university professor mentioned earlier or in the case of the company performing penetration tests at the request of their clients. The background to the general affirmation of the liability for prosecution is that a perpetrator could otherwise make excuses, despite committing a punishable act, that the act performed was in truth not intended at all or can just condemn this sort of activity. In order to get away unpunished after committing the act, the offender would only have to state that in reality he condemns this kind of behaviour. It is clearly obvious that this proposed justification cannot lead to an exemption from punishment.

Despite this, the case of the hackers performing their acts at the request of their customers or for teaching purposes is slightly different. The Federal Constitutional Court of Germany has recently given the all-clear for such cases.

Decisive for the evaluation of any liability for prosecution are, according to the Federal Constitutional Court of Germany, the answers provided to the following two questions: Is the software used a malicious form of software, and were the actions of the "hacker" unauthorized.

Penalty standards are construed such that the offender must have made himself guilty of an offense, for which the punishable actions can be objectively described. Add to this the so-called "subjective offense", which relates to the offender's principal attitude regarding the offense (Jurists define the latter as wilful intent or negligence).

Even if the objective offense of the legal standard is met in as far as the „offender" has used a malicious software product – i.e. a software program „particularly suited" to criminal purposes – in order to spy out or intercept data using passwords or security codes, he is not liable for punishment if the relevant principal attitude required by law is not present. The perpetrator is not considered unauthorized if he uses software to perform simulated attacks on a customer's computer system "on order", which are intended to discover and rectify any security vulnerabilities in the system. The same applies to lecturers at an institute who also do not have any malicious intent.

Giving different reasons, the Federal Constitutional Court of Germany has in effect stated that the activities of the mentioned professor and those of the CEO and employees of the company performing penetration tests are not subject to § 202 c StGB (German penal code).

Improve Quality Services BV

Training and Consultancy

IMPROVE Quality Services

We offer testing and quality management services, including

- **ISTQB** Foundation Certificate in Software Testing course
- **ISTQB** Advanced Certificate in Software Testing courses
- **IREB** Professional for Requirements Engineering course
- **TMMi** Training courses and TMMi Accredited Assessments
- *and many more*

www.improveqs.nl

Special Offer

ISTQB ACCREDITED TRAINING PROVIDER

TMMi® FOUNDATION

contact us now at +31 40 20 218 03
(or info@improveqs.nl) and mention TE4 to get a
15% discount on an ISTQB public course
in The Netherlands or Belgium
(valid until December, 31th 2009)

© iStockphoto.com/TommL

Quality – An agile point of view

by Lior Friedman

In recent years we have been witnessing the agile movement gain momentum. Out of the various principles inlaid in agile methodologies, treatment of the product quality during its lifecycle stands out as one of the key aspects in every agile process. In fact, considering the many innovations introduced by agile practitioners, one can point out quality as a shared underlying theme guiding each and every practice and value used by an agile team.

What is Quality?

From the dawn of software we understood that one of the biggest challenges a software product faces is ensuring that its quality is sufficient enough, sufficient enough to allow the customer to use the product without any difficulties or setbacks and help meet his business needs. But what exactly is quality? Are bugs the only aspect of quality? Should the zero defect goal be the mark dictating whether our job is done?

The answer to these questions is NO!

While unexpected behavior, manifesting itself in software bugs, plays a big role in product quality, it is not the only factor at play. There are several aspects defining quality in a product, in most common definitions of quality we usually find something in the spirit of:

"conformance to requirements" – Crosby

"fitness for use" - Juran

We can even use the definition taken from the ISO standard:

'the totality of characteristics of an entity that bear on its ability to satisfy stated and implied need' - ISO 8402:1994

Another way to look at quality is as "doing things right while doing the right things". But in any case it's not the definition which is important, it's the process in which we ensure high quality and the steps taken to achieve it. From their early stages agile methodologies have treated the process of increasing quality as an exercise in "eliminating waste". (The notion of "eliminating waste" was borrowed from the field of lean manufacturing which talks about all the things preventing a team from reaching its goal). At the end of the day, a high quality product is one that prevents waste of effort invested by the user in his process of producing value, i.e. using the software.

Waste in Software

When talking about quality one can identify three main waste factors that reduce quality in a product:

Bugs

Bugs are the most common causes of waste. Generally speaking, bugs can be categorized as occurrences in which the system behaves unexpectedly. While the developer meant it to behave in a certain way, the exhibited behavior is different due to mistakes inserted during the coding phase. The problem is that when a bug affects the user, in minor cases it might force the user to repeat some action, but in the extreme it can cause a substantial loss of data that will require significant effort to reproduce (if this is even possible).

Usability issues

A hard-to-use product also causes effort to be wasted. Instead of minimizing the work needed, the difficulties a user encounters while operating the system make him waste effort in achieving the desired results. In extreme cases users will resort to various workarounds in order to get their work done. However, in many cases users learn the "wrong" and difficult way to do things, instead of realizing that the actual quality of the system is insufficient for their purpose. If that's not enough, the actual cause of the difficulty is not important. Many developers cling to the notion of 'by design' which in developer jargon is a semi-polite way of asking the user to get additional training on the system. While there are cases in which this is warranted, most of the time it's the failing of the development team to realize that their so-called design is simply not good enough. At the end of the day usability issues reflect badly on the product quality, and like any other waste factor should be eliminated. Remember any feature that requires learning will only be adopted by a small fraction of users.

Unused Features

Capabilities in a system which go unused also represent a waste factor. In fact, unused features are not only useless; they can slow you down and diminish ease of use. However, this form of waste takes more effort to locate and eliminate. Research has shown that in a typical system the majority of the coded features are rarely used. One outstanding example for this is a Microsoft word processor. Usage behavior analysis has shown that 5 commands account for around 32% of the total commands used.

Agile Approach for Improving Quality

The key realization of agile was that the dichotomy between Quality and Development apparent in the industry simply does not work. After years of trying, it was about time to admit that quality is too important to put it solely into the hands of special teams which do not

take part in the building process. In fact, one can say that one of key success factors of agile development is the incorporation of quality into every step and stage in the entire process. This means, that in an agile methodology the responsibility of producing a quality product is shared by ALL those involved in the process and is not the responsibility of a specific group of specialists.

Agile practices are specifically tailored to eliminate all kinds of waste manifesting in a product life cycle, but more specifically, improving the quality of developed product is a major goal threaded into most of those practices.

Short Release Cycles

Agile development can't be called agile unless it is done in short repetitive iterations. When asked what would be the first practice a team should adopt in order to increase quality, most agile coaches will answer 'short iterative development'. When going into short cycles, the pressure on the development cycle increases to the point where most of the hidden factors of waste will not stay hidden. And in order to actually succeed in working in short cycles, most of these factors will have to be dealt with. As a side effect, the feedback cycles will also shorten increasing communication and transfer of ideas which are essential for successful project.

Test-Driven Development (TDD)

Originated in the XP process, TDD is one of the most common practices adopted by agile teams, and yet it's also one of the most misunderstood. Test-driven development is a software design method, not merely a method of testing. Sure, automated suites of test cases that verify the code are clearly a beneficial result, but what really makes this practice stick is the writing the tests BEFORE the actual production code. Automation of tests is as old as software testing itself, although no one can ignore the major impact which the introduction of TDD had on the industry. It appears that writing the tests first results in a better design, cleaner code and forces the developers to actually write them (i.e. test their code). But no matter how it is understood, TDD will decrease the amount of bugs that slip through the coding phase, leaving more time in later stages to achieve better quality. Research shows that if done right, TDD can decrease bugs in a given project by 60-90%, eliminating big parts of the waste caused by bugs.

Acceptance test driven development (ATDD)

Another practice which has gained in popularity with the success of TDD is the automation of the user acceptance tests (UAT). While many view this as an enhancement of the TDD practice, actually writing UAT first has a different goal. By writing UAT before actual work on implementation begins, the business side, with the help of the technical side, is forced to formally define exactly what they want to achieve and specifically how they want it done. Furthermore, the actual writing of the tests turned out to be superior. The resulting test case code was unlike a text document, not open for misinterpretation or for negotiation. After having been written, the goal of the development team was simply stated, make those tests pass and everything else does not matter. Like TDD it was discovered that the benefits of doing ATDD was not merely the resulting suite of automated tests. During the process of defining and writing the tests, both parties get the chance of actually considering what the expected outcome is going to be, and they manage to do it much faster, thus shortening the feedback cycle. The big gain in doing this is in improved usability and better user experience, addressing the second factor of waste and improving overall quality.

Customer Involvement

In the end, however, technical skills and amount of defects are not the only causes for deteriorating quality. Somewhere along the way many development teams manage to establish a relationship with their customers positioning themselves as adversaries. There is nothing more damaging to a project than poor communication between the development teams and the actual users. To address these agile processes usually dictates a heavy presence of the customer, whether by actually bringing a customer representative to sit with the development team (Customer on site – XP) or by explicitly assigning someone to this role (Product Owner – Scrum). No matter how this is done, the results are astounding. Making the development team work closely with a user (or his representative) shifts the focus of the project from technical aspects on how to do things to business aspects of WHAT exactly we wish to do. In the end, no one knows better than the user what exactly he wants to do, and allowing him to state exactly that will reduce the amount of "wasted" features built into the product.

Each party focusses from the start on eliminating unused features. Much more focus is placed on delivering real value to customer- focus on customer cooperation.with short release cycles and early feedback will improve developing only needed features, defects are caught earlier and confusing, hard-to-learn parts will be pointed out earlier in the process.

The only way to go fast is working in high quality

But the most surprising truth agile practitioners have come to realize is that in the end if one wishes to increase development output, the surest way of doing so is by insisting on high quality standards embedded throughout the entire development cycle. In the traditional school of thought, quality was often traded in order to gain a boost in development productivity. However, what many have failed to realize is that bad quality appears to be infectious by nature. Like any disease it can stay undetected for long enough, causing indirect damage which is hard to link to its source. But sure enough, it spreads to all parts of the system causing more and more friction. Only when enough friction causes development to slow down to almost a halt will it get noticed and treated. (This phenomenon is sometimes referred to as "technical debt").

Trying to trade quality for speed is one of the weakest strategies there is. Agile values and principles are all aimed at allowing the highest possible quality standards to be set, knowing that the only fast way to develop a high quality product is to work in high quality mode.

Realizing quality improvement through test driven development: results and experiences of four industrial teams, N. Nagappan, E. Maximilien, T. Bhat & L. Williams, Springer Science and Business Media, Feb 2008.

Biography

Lior Friedman (http://imistaken.blogspot.com) is a Certified Scrum Master and an experienced Agile Coach working at Sela Group in promoting agile adoption and assimilation. After leading the development of cutting-edge testing tools at Typemock LTD and helping numerous companies in their TDD implementation, he provides clients with training, mentoring and high-end consulting services, specializing in AUT, TDD and general agile transitions.

© iStockphoto.com/russlann

A Tester perspective on Test-Driven Development

by Alessandro Collino

Test-Driven Development (TDD) is a core part of the agile process formalized by Kent Beck called *eXtreme Programming* (XP). XP is probably the most agile of the agile processes and it is an extremely high discipline, very effective, and incredibly resilient to change.

That said, it is not necessary to adopt XP in order to practice TDD and benefit from it. TDD is worth doing on its own. Of course, if you are doing XP, it is well worthwhile getting really good at TDD. TDD guides you through the whole life cycle of an XP project. There is no design phase and no explicit testing phase. Both are replaced by automated tests, which are executed continuously to ensure high program quality.

The aim of this article is:

- to analyze, based on my personal experience, the perspective of a tester on TDD
- to identify, in the mental process required by TDD, proper hooks where testers can effectively contribute to achieve better results.

What is TDD?

Some people say: "TDD is not a testing technique, it is a design technique".

Other people say: "TDD is (much) more a design technique than a testing technique".

There is still debate about the nature and purpose of TDD. From a certain perspective it could be considered as a poor debate, since on agile teams using a disciplined process such as XP or a powerful learning framework such as SCRUM integrated with TDD practice, the design, implementation and test phases are not separated phases, but are strictly integrated. However, I think it is important to analyze TDD from different perspectives because of the uniqueness of the approach.

Having joined my first agile team three years ago, and having talked to many developers and testers on several teams, I understand why developers (and not just test-infected developers writing tests first) take completely that former view, whereas testers take mainly the latter view. I am in the latter camp, most likely because I am a tester. I don't think we testers need to claim TDD to be in part a testing technique just because TDD contains the word 'test'. I think we testers are simply more critical (and sceptical), and we spend more time reflecting on our experiences.

Black-box view

When a developer starts to write code, he has expectations about what the code should do. I just said 'expectations about the code' and not 'expectations about the whole system'. Expectations about the code are much more narrowly focused than expectations about the overall system. Let's put those code-facing expectations as inputs on the 'TDD box' in order to get two outputs: the code fulfilling those expectations and a set of unit tests (in a form of an automated test suite that, when executed, will achieve 100% statement coverage). Note that as output I mentioned unit tests. That black-box view shown in Figure 1 (perhaps blind, but I am still reasoning at a surface level) seems to conform more to the following statements: "TDD is (much) more a design technique than a testing technique, but it is in part a testing technique since you get a good set of tests".

In TDD a developer expresses his code-facing expectations using tests. Since many people have doubts about the nature and purpose of the tests used in TDD, I don't call them unit tests. In the remaining article I will use the expression "TDD test" to refer to the type of test used in the context of TDD. So a TDD test is an expression of a code-facing expectation, articulated in advance of writing the code, that prompts a developer to write (just enough) code for the current understanding. That is the essence of a TDD test. In TDD, developers express their expectations in advance of writing the code in a form of TDD tests: this forces developers to be extremely clear about what they intend their code to do.

Now let's put those TDD tests as input on the 'TDD box' in order to get an output: the code fulfilling a set of unit tests as in the previous case. But this time the resulting set of unit tests is viewed as a side-effect. This black-box view (see Figure 2) seems to be conform more to the following statements: "TDD is not a testing technique, but it is a design technique. The set of TDD tests you get from TDD is just a bonus" (but what a bonus!).

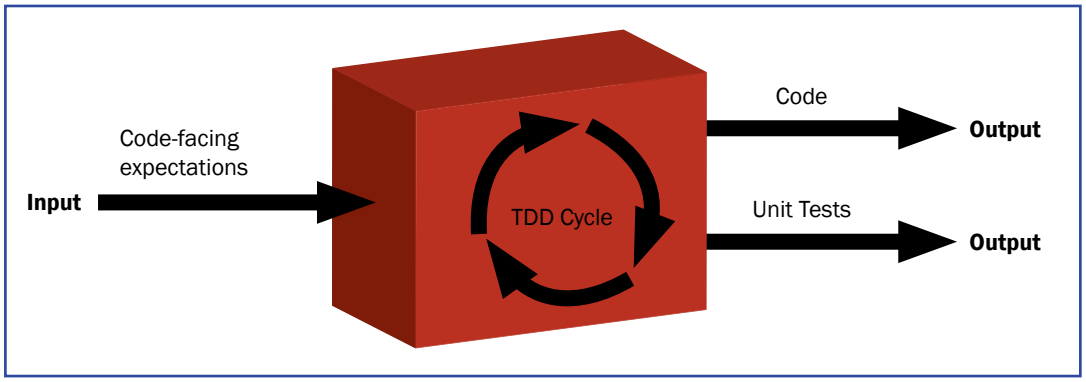


Figure 1

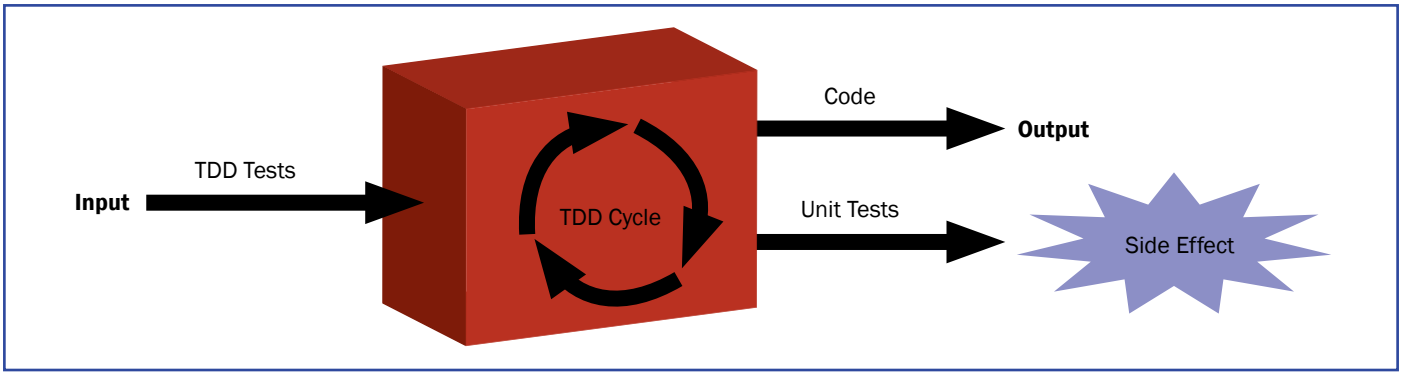


Figure 2

Opening the 'TDD box'

Does it make sense to speak about TDD in terms of black-box testing? Not much, in my humble opinion, since Agile Development is not a phased development where it is easy to identify different complex blocks for each different phase (e.g. Design, Implementation, Testing) and where each phase can be more easily viewed in a black-box fashion. Phased development processes from this point of view are more opaque than Agile development processes. Agile gives visibility, as is brilliantly summarized in Ken Schwaber's phrase: "Agile development will not solve any of your problems – it will just make them so painfully visible that ignoring them is harder".

Let's open the 'TDD Box'. As described above, in TDD tests are written before production of the code, and compilation and testing failures provide the impetus to write production code to satisfy those tests.

A typical TDD cycle follows these basic steps:

1. *Write a test that expresses a code-facing expectation*. This test refers to programming elements that do not yet exist.
2. *Make it compile*. Write just enough code to make the test compile.
3. *Run the test and see that it fail*s. Verify that the test is failing for the right reason, which is that the functionality expressed in the test hasn't been implemented yet.
4. *Make it pass*. Write just enough code to make the test pass.
5. *Refactor*. Refactor both test and production code to keep both amenable to change.

What is refactoring? Refactoring is a disciplined technique for restructuring an existing body of code by altering its internal structure without changing its external behavior. It is a regular part of creating code. Some people understand refactoring as an activity done to fix errors, but that is wrong. A person using TDD refactors when his code meets his expectations and ensures that the product of his refactoring continues to meet his expectations. A person using TDD refactors because the process of writing code changes the way he thinks and understands the code he is writing. Refactoring represents most of the effort expended when doing TDD effectively.

Obviously the TDD life cycle is extremely iterative: a developer having expressed his expectations and then written the code, may then discover that his expectations were not complete. There were other expectations that he did not express explicitly. This is an opportunity to add more TDD tests that express expectations of the code that have already been met. From a developer perspective these new additional TDD tests are considered exactly as the initial TDD tests. From my perspective, i.e. from the perspective of a tester, the purpose is certainly the same, but I think the nature is a little bit different. In the remainder of the article I will use the following expressions:

- *Initial Design Tests* (IDTs) refers to the first tests that are written to yield the initial design
- *Evolutionary Design Tests* (EDTs) refers to the additional tests identified later to drive the evolutionary design emerging from TDD practice.

Both types of tests are certainly about design. In fact, as I just said, the purpose is the same. What about the nature? IDTs are less testing and more checking or, in other words, they are more confirmatory. EDTs tend to be less checking and more testing or, in other words, they tend to be more for fault detection. This is where the tester comes in. I advocate the presence of a tester in a team of people using TDD in order to improve the effectiveness of the EDTs. Based on my experience, a tester that pair-tests with people using TDD can augment in a significant way the effectiveness of the practice.

When people ask me "Are testers needed on a team practicing TDD?", aligned with my vision, I say "They are not needed, but they can help to drive the evolutionary design improving the reliability of the production code".

Another possible idea that could be evaluated is how to obtain a TDD quality improvement based on test design techniques and other testing theories. As a tester I would like to apply this idea, but the question is, can it work? Some people say: "Obviously not, and this shouldn't be surprising since TDD is a design technique and not a testing technique". As tester, and aligned with my vision, I prefer to say: "Hardly, but there are some cases where it can work, and this shouldn't be surprising since TDD is (much) more a design technique than a testing technique". Ok, but which are these cases?

Since TDD is much more a design technique, the quality of the result depends on the quality of the thinking rather than on the quality of the test cases. Put another way, we don't do TDD to get automated unit tests, but we get automated unit tests because we do TDD. Good. What does that mean? It means that knowing more about test design generally will not make a developer better at TDD. Ok, but I said "generally": why?

Knowing how to do a good test design using equivalence class partitioning, boundary value analysis or decision tables can't help to obtain a TDD quality improvement.

Theoretically, coverage from doing TDD is 100% because the developer theoretically doesn't write a line of code unless there is a failing test that demonstrates the code fails to meet some expectation without that line of code. This is the theory, but how about the practice? In practice, there tend to be things that we don't bother writing a test for, so realistic statement tends to be around 80-90% among the TDDers I know (moreover, these percentages refer to greenfield projects, and they are obviously lower in environments with legacy code base). There are a variety of things that the team might do to gain a large increase in code coverage, including paired programming since pairs tend to prevent each other from writing "just this one untested line".

Sometimes, albeit rarely, pair programming alone is not enough. This is where, in some particular cases, a good knowledge of test design could help. In fact to do TDD well, developers have to think through behavior and expectations, and express those expectations in form of a test before writing the code that meets those expectations. So the ability to think through cases and combinations might help with thinking through behavior and expectations, as with state transition testing. Testers are the best thinkers in terms of cases and combinations. Moreover, in environments with a legacy code base, where there is the necessity to retrofit tests onto untested code, testers are more than welcome.

Conclusion

I worked on an XP team both as developer and tester. Based on my experience, I strongly believe that the better way to learn TDD is pairing with a person practicing TDD who has several years of practice and experience in teaching TDD, because doing TDD, and doing it well, is really hard. TDD requires discipline, practice and continuity because to slip back into a "code-then-test" mindset is very easy.

When I was a developer learning how to do TDD, I felt like I was turning my brain inside out. Working as tester on a XP team and having been a developer before, I had the opportunity to evaluate how a tester could best contribute by empowering TDD with added value.

Obviously there are still many open questions. For instance, in the embedded software world and in many industry cases we produce very high-reliability code, but at a cost. Can better development reduce total program costs? I would say yes, but this is still being studied. In embedded software, is TDD applied with rigor going to be better than model-driven development/test? This is probably a good area for research and papers, especially in the embedded software world.

References

[1] D. Astels. *Test Driven Development: A Practical Guide*. Prentice hall PTR, 2003.

[2] K. Beck. *Test Driven Development: By Example*. Addison-Wesley, 2003.

Santander Brazil was twice awarded at eFinance´09:
Software Testing Outsourcing and Software Testing Supporting the Knowledge Management Strategy.

Their partner on both projects is

RSI®

TEST FACTORY
Reduced global costs
Knowledge management
Large scale capacity

TEST ENVIRONMENT
Process definition
Expertise in complex architectures
Data reduction and masking

CONSULTING
Software testing and quality
Change management
Problem management

TRAINING
ISTQB Accredited Training Provider

www.rsinet.com.br

Biography

Born in 1975 and graduated in Computer Science and Information Engineering at the "Politecnico" of Milan, despite of the 7+ years of experience, Alessandro has matured and exploited know-how in conducting various medium-sized and large projects for several companies in various fields, in particular:

- Avionics (working on very complex equipments for the most important Helicopter Consortiums)
- Aerospace (working on a part of the ground segment for the ESA Mars Express project)
- Energy (working in a project conceived to provide new services in the market of energy distribution)
- Industrial (working on consumer electronic products)
- Telecommunications (where he is currently working for innovative platforms for UMTS)

Concerning the above activities, he has worked on all the phases of a complete classical product life cycle, ranging from specification to acceptance testing. In this respect, he has concentrated his efforts mainly on verification and validation activities (e.g. in Avionics he has participated successfully in the certification of an embedded system for a real-time application, according to the DO-178B standard).
He works in Italy at ONION S.p.A. (www.onion.it) and collaborates closely with Dr. Gualtiero Bazzana (CEO of Onion S.p.A. and founder of ITA-SQTB) for important initiatives in software testing.
Alessandro is also active as speaker at Software Conferences, as programme committee member of important conferences, as writer for testing magazines, and is very much involved in the Agile community.

© iStockphoto.com/pavelr28

A way to get better software

by Henrik Andersson

Agile development has during the past years become more and more popular with many organisations involved with software. Unfortunately, however, many organisations tend to forget the QA department when developing software using Agile processes. I have worked as a consultant for many different companies and organisations where they *talked* about how 'agile' they were when developing software. If we scratch the surface, however, their development processes often turn out to be just another classic waterfall where the developer gets a bunch of requirements or features to implement before an end date, usually several months ahead. This gives us long development cycles.

The only real elements of agile in use are the daily meetings in which they discuss what has been achieved, what is outstanding and any impediments encountered. When the developers have implemented the requirements, the software is handed over to the QA department to test the software in a classic manner.

At the end of the cycle there is usually a release, a milestone taking pressure from two sides: On one side are the developers who want their new features in the release, and on the other side is the QA department that is interested in getting the bugs fixed before shipping to customers. Clearly, these two forces do not push in the same direction, which can lead to hostility between the two groups. To complicate the situation further, we then have the product management, marketing and sales pushing to get the new features onto the market as fast as possible, even if this means a reduction in quality. We end up with the business side butting heads with R&D.

This scenario describes a very common way of developing software today.

What is agile?

Before we get ahead of ourselves here, let's take a closer look at the agile manifesto, which was written in February 2001 by seventeen people that needed an alternative to the document-driven, heavy-weight software development processes that were predominant at that time.

What makes a process agile are the following values:

- **Individuals and interactions** over processes and tools
- **Working software** over comprehensive documentation
- **Customer collaboration** over contract negotiation
- **Responding to change** over following plans

One of the principal misunderstandings of the agile manifesto is the part regarding documentation. Many think that agile processes are all about coding and not documenting anything. The agile manifesto says nothing about not writing documents, and many people in the agile community want to restore the credibility lost to the methodology due to this misunderstanding. It is about finding a balance between what documentation to write, and what *not* to write. In many companies large amounts of documentation is written, stored in some repository, and then left where it will often remain untouched, unread and (even worse) unmaintained.

Two of the twelve principles behind the agile manifesto are:

- Working software is the primary measure of progress
- Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter time scale

So we should deliver working software within a couple of weeks if we want to be agile. In short, we have to program and test the new features in very short time scales. Having both the development and testing teams work as one is a must and an accepted fact in the agile world.

The agile manifesto talks about cooperation within the team and that the team should have the flexibility to change direction when required. I have worked with many organisations seeing a connection between the agile manifesto and how they develop their software. But when it comes to testing, the connection is lost, and they stick to having a 'phase' between the point when development stops and the delivery of the product to the customer. This phase is the test phase where the QA team do their 'stuff'.

Classic testing

I am using the term 'classic testing' to make a distinction between agile testing and testing as many of us know it. Classic testing is a process that starts with planning the test, creation of test cases in form of written instructions, manual execution of the tests, creation of test reports, and finally evaluation to see if more testing is needed.

This process can be started as the project starts up, but often has no or minimal involvement with the development process. Rather, it forks off on a parallel track to the development.

Why doesn't this kind of testing work in a true agile project?

Let's take an example. In an agile project there is continuous integration, and we get a new piece of software every morning when we come to the office ready for being tested. If we are to follow the process taught by the ISTQB, the first thing to do would be to plan the test. After you haves planned, you will come to more insight regarding the test coverage. Are the existing test cases enough? Do we need to design new test cases for the new functionalities? If you decide to design new test cases, you have to do that before starting to test the new version of software. Please don't forget your regression testing, so that you know

whether previous features or requirements still work.

As the software grows, both in complexity and number of features, regression testing will consume more and more valuable testing time. To keep up the same pace as before, you can go to your manager to get other people from the team to help you, but this will slow down other parts of the team.

As my beloved testing friends try to say that we don't need to run the regression tests each day, run them as a last overall test at the end of each sprint. So don't we then have a classic waterfall, where we do development and then test?

By testing this way, you will have a long start time before you can start to test and a lot of overhead time in which to plan and design your test cases. However, we want to have as short a start time as possible. This cannot be done by this approach of testing.

How to test in an agile project?

Now finally the big questions: How can we make testing agile and what is agile testing?

The first thing, from my perspective, is that testing has to become a natural part of an agile project and not just a phase after the developers have stopped coding and before customers receives their software. It's about reaching the same goal as a team, i.e. to deliver working software within as short a space of time as possible.

With this short time scale, we as testers need to change our working methods. We need to step away from the nagging position we have in a waterfall process and become more active in the process of developing the software. Active does not mean that we should be sitting down hammering in code, but we should be involved from the beginning and be giving the developers feedback instantly on how the new feature works, instead of complaining at the end when everything is built.

To make this possible, we need to have frequent integration of new features, for example through nightly builds, where every morning there is new software ready to be tested. The testing should be performed manually as an exploratory test, since this is a rapid way of testing instantly, no extra documentation, no extra work, just the minimum effort required to succeed, namely testing the new features.

But how can we possibly do regression testing when we have no test cases?

When I talk to fellow testers who are into classic testing, we soon run into disagreements:

- We need the test cases for regression testing later on when the product has come in to a maintenances phase, they say.
- Let's spend time to automate these test cases, so we don't need to spend time running them manually later on, I reply.

Let me explain how I see things. Assume that we have a software project that is at the beginning of its life cycle. Not too complex, the GUI has started to fall in place, let's say it is a web application. We need to start to test automatically as soon as possible, which can be easily achieved using a tool that records our clicks and keystrokes. After each nightly run of the automated tests, we then add new test cases the next morning. This means that we get some extra time to add new automated test cases, or do some exploratory testing, since we don't need to test the existing ones manually.

The profile for an agile tester is not the same as the profile of a classic tester. I see the agile tester as a person with a more technically orientated profile, a person that enjoys both developing, debugging and testing. This profile is wanted in agile projects, since it will help the developers to find where the bug is, instead of just pointing out the symptom of the bug.

In summary, in agile testing you need to start to execute tests fast. Having working software early is a key aspect, and this means you have to have testable software early. Exploratory testing is ideal for you to be able to make a fast evaluation of the software with a minimum of preparation, and experience has shown that it is also very effective for finding problems in the software. Critical parts of the software should never fail, and you need to make sure that they are not broken in the fast development cycle. Because of this, it is worth putting effort into implementing automated regression tests.

As an agile tester you must be able to tackle these things, which calls for an interest in technology and a deeper technical understanding. Programming and debugging are key skills for the agile tester.

Biography

Henrik Andersson is ISTQB Certified Tester and test consultant at Testway, Malmoe, Sweden. His main focus is on automated tests in agile projects and on exploratory testing. After studying software engineering at the University of Skovde, Sweden and writing his diploma thesis on low-level testing, he started as a C++ developer at Tern System, Iceland. Henrik is also one of the founders of a test community in southern Sweden. In his spare time he loves riding his mountain bike and dancing "Lindy Hop" (better known as "Jitter Bug").

File: test_reports_and_screenshots.zip Size:1.67 GB

100%

Recipients: John M., Edward J.

SEND COMPLETED!!!

Send smoothly with
www.bitzen.net

bitzen.net

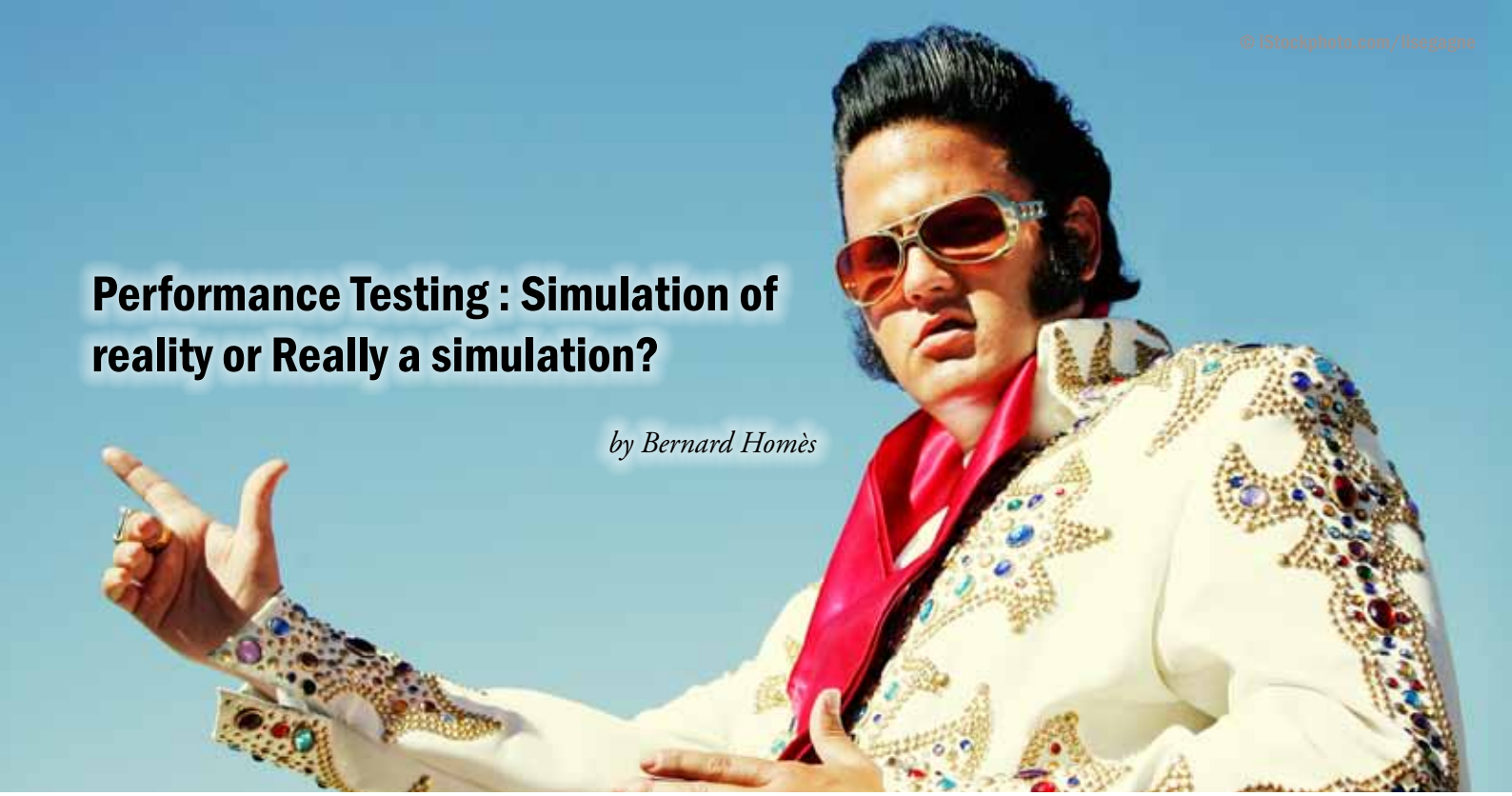
Exchange files
with customers, suppliers and partners in
a simple, fast and secure way.

Specifically created for companies and professionals.
Plan free.
Monthly plans starting from EUR 6.00
Paypal.
Send large files up to 2GB.
Batch files upload.
Unlimited contacts.
Automatic alerts for sent and uploaded files.
Resend files.

No additional software installation needed (SaaS).
No Virtual Disk, no FTP and no Webmail.
Supports any kinds of files.
Compatible with IE, Firefox, Safari, Opera and Chrome.
Automatic daily backups.
No advertising. No Spam.
No public links.
Unlimited number of downloads.
Secure SSL.

Web: www.bitzen.net Email: contact@bitzen.net Twitter: @bitzennet

powered by amazon web services



Performance Testing : Simulation of reality or Really a simulation?

by Bernard Homès

Different aspects come into play when executing performance (load) tests of servers: one is the server load, the other is the user experience. These two major aspects have to be measured in order to ensure that the results are within the expected boundaries.

This article has two parts. The first part focuses on how to ascertain that user experience is correctly simulated.

Definition

Performance testing can be understood differently depending on the context. In these articles we consider performance testing to cover the response time (as experienced by the user) of servers when subject to transactions from a large number of simultaneous users (local and/or remote) and performing normal operations. Normal operations are considered as 24/7 operation with back-office operations (i.e. batch processes) occurring simultaneously.

Transactions can be any type of request from users, from simple page requests to full online purchases.

Realism is understood as being the ability of the performance campaign to accurately reflect what will happen in the field with the number of simulated users. This includes the impact on the server and the response time experienced by the users.

Introduction

Performance testing includes aspects on the server side and on the client side. The aspects on the server side will be described in the next article of this series. The client side aspects will be developed here.

Client-side aspects are not limited to sending a string of requests to the server and timing the responses, but include a verification of the client's response in itself (i.e. user experience). It is thus necessary to set up a multi-tier test automation strategy.

Application types

Applications which are subjected to performance tests can be of different types. We can have (1) terminal-type client, where there is no processing on the client side, (2) thin clients where the processing is limited, and (3) heavy clients where computing is distributed between the client and the server. This last type of applications leads to more complex transactions between client and server.

There are also severs (or applications) constructions that make frequent usage of pop-up windows and/or secondary windows.

Automation strategy

A performance test automation strategy aims at testing the user experience when the server is subjected to a large number of user requests over a given period of time. The performance aspects are important to evaluate as they impact strongly on all the aspects of the software development. Lack of performances will most likely have a large financial impact on software development costs and duration.

The automation strategy will cover the selection of automation tools, as well as the definition of the transactions selected.

One should not forget the principal reason for testing, which is finding differences between expected and actual results and providing reliable information to management.

Finding differences requires that expected results are known and can be checked against actual results during test execution. Providing reliable information requires that the reporting presented does not lead to incorrect interpretation (it might be better to provide pessimistic instead of optimistic evaluations, in order to ensure future positive user experience).

Transaction selection

When one selects the different transactions to simulate, one generates some deviation from reality. If the transaction selection does not match the load the server will experience, the performance campaign will not realistically simulate the reality. Thus transaction selection is very important in performance tests.

There are transactions that are more resource-consuming than others, there are requests of simple display pages and dynamically created pages. All these generate different loads on the server, and it is necessary to ascertain that the load generated by the clients correctly represents the spread that will be found in production.

The challenges

#1 Create a valid load (Load validity)

The first challenge is the selection of a subset of different transactions from the set of all transactions that can be executed by clients. The spread of transactions, and the percentage of each transaction in relation to each other is of the essence here. Whenever there are back-office transactions that have to be taken into account, it might be necessary to verify the impact of their execution upon the response time experienced by users (more on this in the second article in this series).

#2 Create realistic users (User Realism)

The second challenge is to have the users behaving as realistically as possible, and thus generating as realistic a load as possible in terms of spread of data within a single transaction (i.e. not always looking for the same set of

data in a table).

#3 Validate the responses (Response validity)

A standard practice for testing is to test the actual results against expected results. In performance tests, what often happens is that the transaction responses are timed. This relies on the responses being of adequate quality (functionally). This is contrary to a basic tenet of software testing that implies that you cannot rely on an answer to remain the same if conditions change. When one is stressing a system, such a reliance is unwarranted, and the response validity is to be verified.

#4 Reduce the number of injectors [1](Injector reduction)

Simulating a large number of clients is the fourth challenge. Here a fine balance exists between adding more machines and simulating more clients per machine. The first option has an impact in terms of hardware (adding injectors, connecting and synchronizing them during campaigns), while the second has an impact in terms of load on the client (injector) machine, as simulating a large number of transactions and validating the responses obtained creates a load on the client hardware.

#5 Transactions coherence (Transaction realism)

A fifth challenge comes in terms of the validity of the stream of transactions coming from a client. This is the case when the transactions involve different screens linked to each other, such as when purchasing an item from an online store, or processing banking transactions. This requires the server to keep data linked to the opened session through means other than cookies on the client side or inclusion of the full information in the request [2].

#6 Encryption (Encryption impact)

The use of virtual private network (VPN) and dynamic encryption methodology to ascertain that the transactions are secure can have an impact on the ability of capture replay to create transactions that will be correctly interpreted by the server and not rejected due to invalid encryption keys (that would happen in case of encryption based on keys synchronized between server and client).

Type of injector tools

Injector tools are necessary to create transactions in large numbers and monitor their responses. You might wish to have a look at the sidebar to understand how a transaction is created. There are different ways to generate transactions from an injector: through scripts, and through capture/replay tools. There are different capture/replay tools and we will differentiate between the 'classical' tools and those embedded in internet browsers.

Script-based injectors

It is relatively easy to create a script (using any scripting language) that will generate hits on server addresses and compute the time difference between the moment the request is sent and the time the server sends information to answer the request.

This type of injector is of course unable to create coherent transactions and received data is difficult to verify. On the other hand, such a script-based injector is very good at generating a large number of requests on single (i.e. non- related) pages. It can be used whenever a background load has to be generated.

Classical capture-replay tools

Capture-replay tools are software tools that simulate the action of a browser by recording user actions and then replaying these actions automatically. It is generally possible to modify the content of variables used, so that challenge #2 (*User Realism*) can be covered. To be able to replay transactions correctly, capture-replay tools rely on their ability to decode the actions executed by the user on the software components in the page. Whenever new types of components and protocols are used, the capture-replay tool vendors have to update their tool to take this into account. Therefore there is always a delay between the introduction of a new type of component and its ability to be captured and replayed.

Capture-replay tools can often manage cookies stored on the client's disk. Cookies kept in memory (i.e. never written on disk) are less often managed by capture-replay tools.

Capture replay tools contain means to verify that the reply obtained conforms to expected results. Of course this is only valid for components that can be recognized by the tool. There are some components (such as Flash, Shockwave and applets elements) that are opaque to the capture-replay tools. They can thus not be verified and their content is not managed by the tool.

Browse- embedded capture-replay tools

Classical capture-replay tools are written using normal programming languages and emulate the way a browser works. If built on top of an existing browser, a tool is able to benefit from all the technology present in the browser (i.e. anything that the browser recognizes can be recognized by the capture-replay tool) and thus process coherent transactions (sessions are maintained by the browser), manage opaque components (as can the browser) and use the multi-threading technology of the browser. The capture-replay software can monitor and use any protocol recognized by the browser.

Response validation

Responses provided by the server to transaction requests can be of different types: valid or invalid. Invalid responses as recognized by the browser include all the 400 series HTTP errors (i.e. 404 no such page, …). These errors are easily detected by capture-replay tools. Some errors, however, can be recognized as valid by the browser, but considered invalid by the user (i.e. "transaction not allowed", "transaction declined by your bank", …). These errors, while correctly detected in the functional tests phase, are not verified – by most performance testing software – during performance test execution. When a server declines a transaction during performance testing, the response to the transaction will most likely be shorter than what is expected and can be misinterpreted as valid instead of invalid.

Depending on the type of transactions, the lack of response validation can lead to unrealistic response times being computed.

User experience

Users experience what is displayed on their screens, not what is received by their machine. In case of thin clients, the time to display the information is negligible. For heavy clients, where there is a lot of computing on the client's side, the time to display the information becomes more important. A performance tool that can provide the time from request to display will provide more accurate (i.e. realistic) values than one which times the receipt of the last TCP packet.

Encryption and security of transactions

A quick look at news reports shows that information (i.e. content) security is of paramount importance in today's business world. Encryption algorithms (PGP, RSA, …) and the use of VPN can have an impact on the response time of transactions between clients and servers. Some capture-replay tools support secure protocols such as https, others do not. Secure security algorithms can be time-dependent (the user is provided with a separate hardware device that contains the encryption algorithm and delivers encrypted keys used to sign the transactions), and this makes replay of captured transactions impossible (i.e. the server will correctly receive the request, but will not be able to interpret it –wrong key timestamp–, nor provide adequate answers –due to invalid key–).

To deactivate the security algorithms during performance testing is one way to overcome this, but leads to another area of uncertainty, another lack of realism. A partial solution to this problem would be to let the encryption algorithms work on the injectors, and process the transaction only at session or GUI level.

Comparisons

We have seen that different tools are available (script- based injectors, classical capture-replay tools and browser-embedded capture-replay tools) , that under the same concept of server there are applications of varying complexities, and that the choices made during performance campaign preparation have impacts on the realism of the tests executed.

We can thus see that context-dependent choices are to be made, and that they have a large impact on the realism of the simulation.

Each choice made should be analyzed in terms of impact and the risk associated with it. A

1 Injector : equipment (hardware and software) that generates the transactions in a performance test.

2 This method of passing data between client and server is prone to security attacks.

way to compute the resulting realism (or lack thereof) should be implemented to ascertain that the sum of all risks and impacts is not above an acceptable threshold.

The following example can help to understand this suggestion. We will weight the risks identified and determine if the uncertainty is in line with the expected accuracy of the performance tests.

Risks identification, evaluation and weighting [3]

Risk #1 : Number of selected transactions per number of possible transactions. If 75% of all transactions are taken into account, and this represents 90% of the expected load, the estimated realism could be computed as: 75 x 90 = 67.5%

Risk #2 : Number of different values within the same transaction type. Each transaction requesting database information should select a different record (realism 100%) or select the records from a subset of records (realism 20-80% depending on the size of the subset and number of identical transactions). The complexity of simulated transactions will reflect the real complexity of actual transactions.

Risk #3 : Opaqueness of the components (http / https / flash / shockwave / …) has an impact on the ability of the capture/replay tools to simulate the interactions with these components. Components are sometimes opaque and the capture-replay tool is unable to process opaque components: realism 50%. If such components are not used or the capture-replay tool can process them: realism 100%

Risk #4 : Type of server pages (server architecture and technologies used). Static and dynamic pages, as well as multi-tiered architecture and how they are rendered in the test environment have impacts on the performances as seen by the users. If commonly used pages are stored on page servers, the response time experienced by the user will be faster than the response time of the real server. The impact on realism has to be determined.

Risk #5 : Applicability of the capture-replay tool used to the pages / transactions selected. This relates to the applicability of the tool to reproduce the way transactions are measured and managed.

Risk #6 : Presence (or absence, or removal) of encryption technology, type of encryption technology (i.e. time-dependent of fixed key, hardware or software based). The impact of encryption technology and all security measures (firewalls, etc), even though minor in themselves, is not negligible and should be taken into account. This impact can affect the transaction times and the server loads. Realism if transaction security is implemented: 100%, if not implemented suggested impact on realism: 90%.

Risk #7 : Type of cookie, ability of the performance tool to manage the cookies (session based cookies, permanent cookies, …). The type of cookies used by the application under test will have impacts on the test setup (for permanent cookies, is the cookie already present?) and on the way the transaction is managed. If the test tool manages the cookie, does it manage session cookies as well as permanent cookies? The impact of cookies is important when simulating multiple users with a single injector. Each session in the injector should represent a separate user, with its associated permanent and session cookies. However, if permanent cookies are used, the first session downloads the cookie and the other sessions will not have to download the cookie, leading to lower transaction load than would really happen in the field. Impact on realism: cookies fully managed at session level (both permanent and session cookies): impact 100%, session cookies used and managed (not at permanent cookies): impact 90%, permanent cookies used and managed: impact 80%.

Risk #8 : Number of simultaneous virtual users on an injector. This risk is inversely related to the realism of the test tool since a simple tool (i.e. with no built-in logic, and few checks available) will be less costly in terms of CPU and memory usage per simulated virtual (synthetic) user than a more powerful tool that more realistically simulates user activities. The number of simultaneous users is often seen as one way to reduce the cost of performance tests, as less injector machines will be used to simulate the same number of users. This is at the cost of realism.

Risk #9 : Estimated bandwidth required vs. available bandwidth. When creating a test setup in-house, the available bandwidth is large and there are few bottlenecks. This will enable many simultaneous transactions to be simulated, but not realistic simulations. Transactions made over dial-up lines and at lower speeds ,(that still represent a large part of available Internet connections), are often prone to packets which are resent and connection problems and will not be realistically simulated. What's more, the input bottleneck (connection of the server networks to the outside world via ISP or dedicated lines and associated modems) will be bypassed, leading to a significant lack of realism.

Risk #10 : Type of application (thin client vs. heavy client). The results of performance testing also vary if the application is a client-heavy application or if it is a thin- client application. "Heavy-client" applications place a larger load on the client side, so that the client performs part of the processing. This means that the transactions between client and server(s) are not simple URL requests, but can include SQL requests (send a query, receive a list of records) and other specialized transactions with specific protocols. This kind of "heavy-client" application is not the usual application on web servers, but becomes more and more common to take off some load from the servers. This is a challenge for capture- replay tools that are not based on web browsers. Thin client's realism can be estimated at 90% while users with processing done on the client side have a realism that is dependent on the capture-replay tool's technology, and can range from 40 to 90%.

Challenges coverage

Challenge #1 (*Load validity*) can be covered by a selection of adequate transactions. The selection can be based on expected usage (provided by marketing) or by an analysis of actual transactions (based on a previous version of the software or on statistical data). The number of different transactions simulated has an impact on the work load for test engineers. A balance between the number of different transactions simulated and the level of realism achieved must be reached. The more realistic the spread of transactions, the more realistic the results will be.

Challenge #2 (*User Realism*) can be covered through two complementing ways: one by generating enough data in the tables (see the part about database population in the second article in the series). The other way to cover challenge #2 is to have test data dynamically accessed during the execution of the performance test campaign.

Challenge #3 (*Response validity*) can be attained by checking the actual responses (from the server) against expected responses. This means that the injector tool must have a validation capability. Another way would be to add a set of functional tests – where the content is validated – during the execution of the performance test campaign.

Challenge #4 (*Injector reduction*) is a compromise in terms of realism between the number of injectors necessary to simulate the load on the server and the lack of realism that is due to the different types of injectors (script, classical, browser-based), the selection of transactions executed, …

Challenge #5 (*Transaction realism*) can be obtained through session-based information (if the performance tool manages that) on the client side. It is not possible to have session coherence with a script-based injector, and it is difficult to ascertain this with classical capture-replay tools, as there is no guarantee that the sequence of transactions generated are all executed within one session.

Challenge #6 (*Encryption impact*) is often not taken into account during performance tests. The reasons mentioned are that the encryption aspect is (should be ?) covered during the functional tests, not during performance validation.

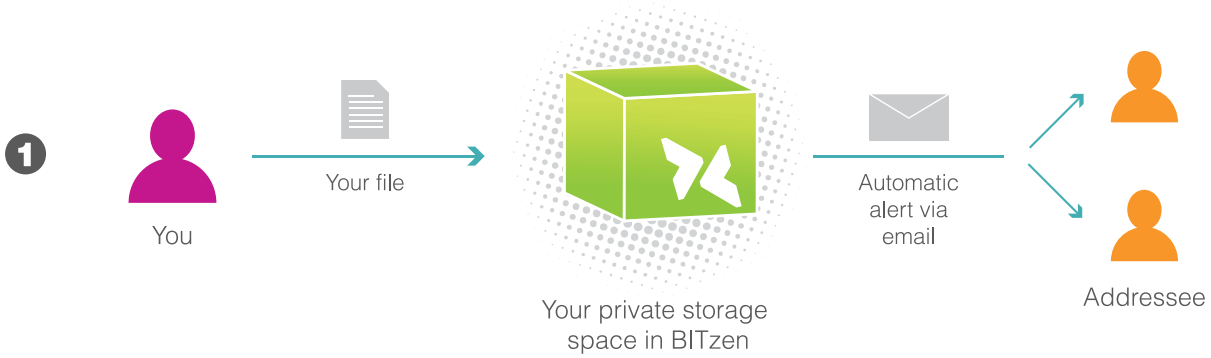
Challenge #7 (*Load balancing*) is often difficult to take into account during performance tests. The reasons are that the test environment cannot be faithfully reproduced, and that the balancing between the servers is not always reproducible during the performance test campaign.

Application evolutions

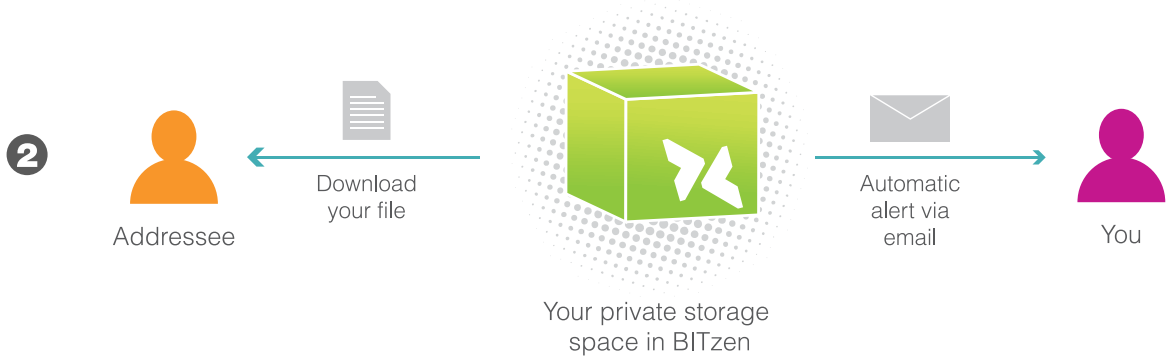
Evolution of the software tools leads to evolution of the type of applications being developed. We have come from static character-

3 Risk occurrence and weighting are to be adapted to your own projects, these values are only examples.

Send smoothly...



Upload your file to your private storage space in BITzen.
BITzen inform your addressee via email that they have recieved a new file to download.



Your addressee downloads your file over a secure SSL connection.
BITzen informs you of the successful download.

...with www.bitzen.net

bitzen.net

Exchange files
with customers, suppliers and partners in
a simple, fast and secure way.

- Specifically created for companies and professionals.
- Free account with 100 MB storage space.
- Monthly plans starting from EUR 6.00.
- Paypal.
- Send large files up to 2GB.
- Batch files upload.
- Unlimited contacts.
- Automatic alerts for sent and uploaded files.
- Resend files.
- No additional software installation needed (SaaS).
- No Virtual Disk, no FTP and no Webmail.
- Supports any types of files.
- Compatible with IE, Firefox, Safari, Opera and Chrome.
- Automatic daily backups.
- No advertising. No Spam.
- No public links.
- Unlimited number of downloads.
- Secure SSL.

Web: www.bitzen.net Email: contact@bitzen.net Twitter: @bitzennet

powered by amazon web services

based pages to dynamic pages embedded with multimedia components, including both streaming audio and/or video data. The shift to broadband and ADSL internet connections enables these heavier data loads to be provided to the clients, but they place a higher stress on the servers. We can predict that in the future we will see (1) more and more complex pages with new types of embedded (opaque) components; (2) more complex transactions where data will be cached in the client's memory (and not on its disk) and/or on the server's; (3) more complex and heavy client side processing as is possible with AJAX and (4) a race between the test tools vendors to catch up with the new protocols and/or (opaque) components.

Synthesis

The different challenges mentioned in this article are probably not the only ones facing performance testing on the client's side. The aspects on the server's side will be dealt with in the second article of this series.

What is the goal of performance testing? To provide a simulation of the real workload facing servers, with as high a degree of realism as possible. Any other goal is a waste of time and resources.

Performance issues are those that are the most difficult to solve. They are usually evidenced when the site is up and running for some time already, when the marketing campaigns have already been played, when the customers are many and eager, when the costs of development are already quite high, and when the time to fix things is much tighter. The impact of performance issues is very high in terms of visibility (to a large number of customers), in terms of time to fix (we cannot afford to anger customers for a long period of time) and in terms of costs (we might have to rethink the architecture of the application). Performance testing is used to avoid these risks. To accept less than 100% realism in performance testing is to leave oneself open to defects that will only come to light when a large number of users will connect to the system.

Conclusion

Actions can be taken to maintain a high degree of realism in the simulation, whether through transaction selection, and through analysis of the application. However, one item that is often overlooked is the type of tool used: is it a simple script firing URL requests to the server, is it a more sophisticated capture/replay tool that cannot process all the protocols, or is it a browser-embedded tool that can reproduce all that a browser can interact with?

This last type of tool requires more resources on the injector's side, but with the guarantee of higher realism. And what's the cost of a few injectors compared to the losses if the system (hardware plus software) has reached its limits before the expected time ?

Second part

Different aspects come into play when executing performance (load and stress) tests of servers: one is the server load, the other is the user experience. These two major aspects have to be measured in order to ensure that the results are within the expected boundaries.

The first part of this article covered the aspects from the client's (injector) side. The second part covers the server evaluation and the different aspects that should be taken into account on the server side.

Introduction

Many articles and books have been written on performance testing, covering topics ranging from tools available to processes, from economic justification to methodology for performance testing.

A large number of assumptions are made during performance testing preparations and executions. These assumptions have an impact on the realism of the test execution and thus on the adequacy of the results to reflect real world performances.

The challenge

Whenever performance issues crop up during the lifetime of the software application, the standard answer is to upgrade to a more powerful version of the hardware. This, however, is often not enough and only serves as a temporary stop gap measure.

Performance testing issues are multiple, as many different causes may impact on the performances (a comparison can be made to connecting pipes of different diameters with funnels and expecting the same rate of flow). The challenge is to determine exactly where the performance issues (bottlenecks) arise, and their impact on the flow of data from client to server and back to the client. The aim being to create a test environment (on the server side) as close as possible to real life, and thus obtain results that are in line with what will be experienced in the field.

Server organization

Multi-tier server organization impacts the response time. This can be expected, as the application resides on a server or a cluster of servers, and their connection will impact response times.

A large number of connections between the different servers will have a negative impact on the application's response time. It is necessary to ensure that the test setup corresponds to the actual (i.e. production) setup. Any other architecture will be less representative and lead to results that do not correspond to the live environment.

Database population and content

When testing an application that makes use of a database, the standard tests target a limited number of records stored on the database. Database organization is based on a number of index tables and actual records. The number of records has an impact on the number of index tables and index levels. When the database is filled (i.e. after some weeks, months or years of operation), the total database record access time will increase, as the number of index tables to read will increase. The index tables contain pointers to other index tables and to actual records.

Database server optimization often includes caching data that has been recently and/or frequently retrieved in order to avoid re-accessing the actual database record. This enables a quicker response time whenever a record is accessed many times. In test cases the number of records used will of course be limited.

In order to avoid the server caching data and to force the server to effectively retrieve the data each time, it is necessary to be aware of how the DB server stores and retrieves data, and select test data accordingly.

Number of reads

Another aspect linked to this concerns the size of the data sets available (number of records or items), whether the data is in the form of files or data in a database. The number of records read before the correct record is retrieved depends on the number of index layers, the size of the blocks and records, as well as on the number of records in the file, the size and sepa-

Analysis of a transaction

An internet transaction starts when the client sends the request to the server, and ends when the server(s) have provided the full answer. Reception of the 'full' answer depends on the type of components included on the page (i.e. required by the client). When you load a page from a server, you may notice that at the bottom of the browser an area displaying the text "completed", while at the same moment some components on the page are not being displayed properly, or a "loading..." message appears on some components). Both pieces of information are correct: the page is complete in terms of display of the layout, but a separate thread or process has been opened to download the specific software component and that download is not yet completed. This is visible when loading Flash and/or Shockwave components, and can also occur when loading images and embedded music or movies.

Whenever one connects on a live streaming web-radio, this is something that also occurs.

Another aspect that is often present is the generation of multiple parallel answers from the server to the client and assembly by the client of the displayed page. This means that the client has to determine when the page is complete and no further data is expected. In such a case, the reception of information from the server stating that the data has been totally sent might be misleading.

© iStockphoto.com/pidjoe

ISTQB® Certified Tester-Foundation Level
in Paris, in French

Test&Planet : votre avenir au présent.

Venez suivre le Cours « Testeur Certifié ISTQB – Niveau fondamental » à Paris, en français !

Test&Planet, centre de formation agréé DIF, vous propose ce cours D&H en exclusivité.

Apprenez à mieux choisir Quoi, Quand et Comment tester vos applications. Découvrez les meilleurs techniques et pratiques du test logiciel, pour aboutir à une meilleure qualité logicielle à moindre coût.

Possibilités de cours sur mesure.

Pour vous inscrire ou pour plus d'information : www.testplanet.fr

Díaz Hilterscheid

Test&Planet

PARASOFT
We make software work.

Quicker
with the full automation of time-consuming repetitive tasks you reduce time to market.

Better
with innovative technology you achieve better product quality to strengthen your competitive advantage.

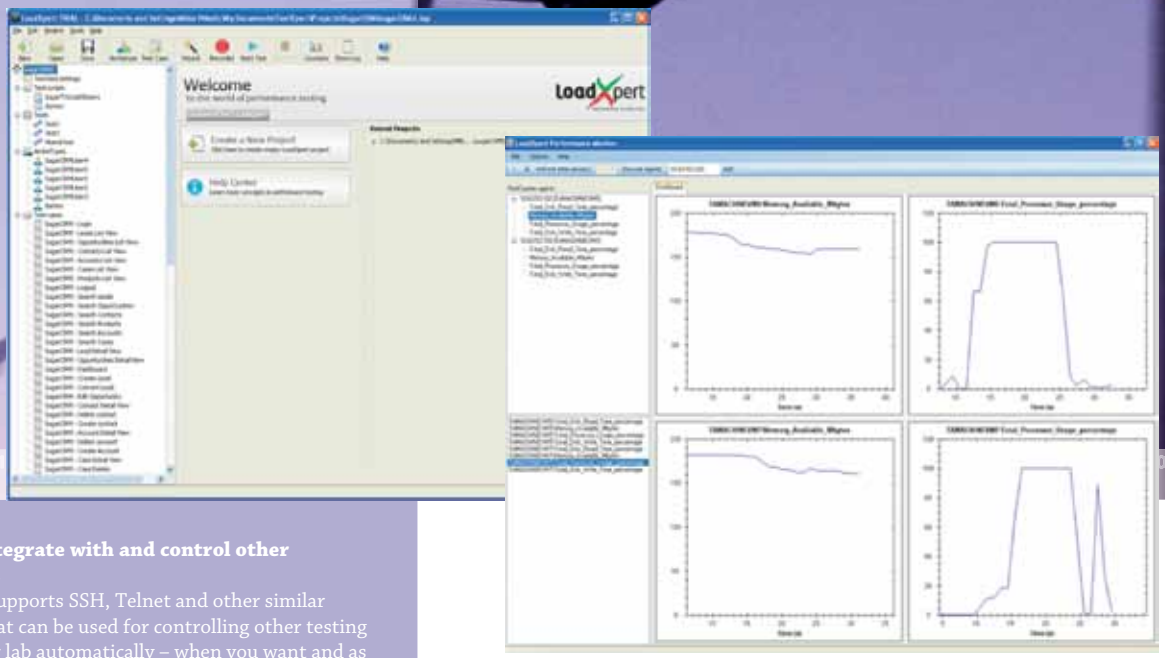
Cheaper
with the most cost-effective technology you significantly reduce operating costs.

phone: +44 (0)208 263 6005, e-mail: sales@parasoft-uk.com, www.parasoft.com

Low hardware requirements, scalable system with variable number of load generator machines.
System is a client-server application, where clients are controlled by the server. Server dispatches the load generation tasks onto as many clients as you wish, one client can be installed on one machine. You can use whatever PC machines you have in your LAB.

Easy to use script recording process
Preparing your test scripts has never been easier with our one-click-recording process. You can start the recorder and then continue using your application as you would normally do. LoadXpert will capture all the traffic that goes between your browser and your application. Later that traffic will be parameterized and multiplied onto as many instances as you want.

Test your applications Just as your clients do. And do it affordably.



Ability to integrate with and control other testing tools
LoadXpert supports SSH, Telnet and other similar protocols that can be used for controlling other testing tools in your lab automatically – when you want and as much as you want. Let LoadXpert be your main tool for your complex performance tests.

Flexible and easy to use load design wizard
After you have recorded your test scripts, you can easily design the load that you want to throw at your application, by using LoadXpert load design wizard. You can choose how you want the load to reach its maximum and then how long you want your load test to last, by following on-screen instructions in our wizard.

Server performance monitoring system, which monitors windows/linux performance counters in real-time.
With LoadXpert Monitor you can keep track on your server performance in real-time and record all performance measurements into a database for later processing. LoadXpert Monitor supports both Windows and Linux based servers and applications. It can be used prior, during and after load tests.

Fast growing happy customers list!
Proven tool for performance testing of Web Applications (Cisco and Hypo-Alpe-Adria bank as customers)

- Integrated Development Environment (IDE) for test script editing and traffic recording
- Load Script Wizard, for quick and easy way to design the load scenario during the test
- HTTP/HTTPS Recorder, simple utility within IDE used for capturing HTTP and HTTPS traffic between users and application
- Scalable Client-Server architecture, for emulating concurrent user workload, where clients are controlled by the server. Server dispatches the load generation tasks onto as many clients as you wish, with one client software per machine.
- Scheduled Test Execution, you can schedule your tests, to start at any given moment in time (overnight testing etc)
- Real-time Server Performance Monitoring System, for monitoring and recording server performance by collecting performance counters from all underlying hardware and software servers (both Windows and Linux servers are supported). Performance counters are recorded to a database, for later detailed analysis. Web viewer version available also.
- Automatic Report Generation, test execution report with graphs and data analysis results is just a few clicks away.

- Performance testing tool offering features of contemporary products
- Low hardware requirements tool for SMB and enterprise sector
- Easy and intuitive to use environment with quick test preparations
- True simulation of real-life load using archetype methodology
- Quick test reports generation with more time for results quality judgment
- Variability and scalability of test process
- Predictability and productivity for performance testing professionals
- Return on your investment based on your quantitative results

Contact us for more information, trial version or try-out option at
http://www.loadxpert.net

LoadXpert
PERFORMANCE TESTING TOOL

ration of the different blocks of data. This is visible whenever a disk is defragmented (response time is better), and is applicable to all types of disk storage as well as in database management engines.

The impact on the response time comes from the number of accesses to the disk necessary before the correct data is retrieved and provided to the application.

Static pages

When the same (static) page is frequently requested from a server, the server may elect to keep that page in memory, thus providing a faster response time than would occur if the page had been retrieved each time. This can occur when executing performance tests, as the variation in pages requested might not reflect the spread of pages that will be requested in the field.

The impact is not negligible, but is seldom taken into account. To minimize the impact, the static pages selected should reflect the curve that statistically represents the pages requested by users.

Dynamic pages

More and more of the pages displaying up-to-date information are generated dynamically: train or plane schedules, time-specific offers require information that is time-dependent. This of course makes use of the data provided in databases and other application servers. The size of the page may be relatively small in terms of bytes transferred, but the remote servers (the different tiers of a multi-tiered application) must be able to provide the information that is to be displayed on the dynamic page. To simulate this during performance tests forces the test environment to be as close to the production environment as possible. This is often not reflected in the resources allocated to testing, leading to inaccurate data.

Cache and memory

Just as on the client's side, pages of data that are frequently requested can be stored on the server's cache memory. This allows the server to provide the data to the user without physically accessing the data on the support media, thus speeding the response time. This is often the case when a limited set of data is used in performance tests, or when the data storage media has a large reading cache. Whenever performance issues arise, enlarging the data cache is a common response.

Another aspect related to memory and cache is linked to multi-processing: whenever a process is loaded, data and instructions are loaded simultaneously. The processor allocates this based on the amount of physical memory available, and on the amount of cache allocated. While this is common knowledge, its impact is often not taken into account when executing performance tests.

Threads and Processes

Back-office processes (batch updates and other processor and/or I/O intensive processes must be included in the selection of activities done at the same time as on-line processing. This includes the creation by the main application process of multiple threads and processes. Any new process will include memory allocation (see Cache and memory), and will take a small share of the CPU usage. The more processes, the more the system will swap between processes, attempting to share a limited resource (time) across a larger number of customers (the processes), resulting in smaller shares for all. In extreme cases the system can waste all its time swapping processes in and out without accomplishing any meaningful task for each.

Depending on how the application is developed, higher number of users often lead to the creation of a large number of threads or processes, each of which are allocated a piece of dwindling resources. Whenever a threshold is reached (said threshold varying on the server configuration and performances) the resources allocated to system management take a significantly larger part than those for the application being tested.

Other applications

Many pages displayed via internet contain some kind of advertisements, publicity or company-sponsored links. These can be static or the result of some processing based on previously visited pages.

Advertisement and publicity banners request access to other servers, most often not linked to the ones processing the application being tested for performances. Nevertheless, the user's browser is required to access these other servers, sometimes transmitting user-specific data (such as language) in order to provide the information to display to the user.

Performance issues are not limited to what is displayed to the user, but includes the entire purchase / supply chain, such as whenever a purchase transaction is made, stock has to be updated, as well as accounting and the material is to be forwarded to the buyer.

Network

Network aspects might be worth an article all by themselves, let's look at selected aspects: speed of connection, type of connection (throughput and bandwidth), multiple network protocols, and VPNs.

Speed of connection

The last decade or two has shown quick development of networks and networked applications. Introduction of high-speed networks in cities and between continents have reduced the perceived distance between the user location and the server location. Even though telecom operators have invested heavily in optical fiber and other equipment to provide ever faster communication, any advance is quickly overtaken by the higher volume of data being transmitted. One should also remember that advent of T1 lines does not mean that older generation modems are replaced immediately, nor that copper cables are replaced overnight by optical fibers.

This has a strong impact on performance testing, as two identical transactions (including identical user think time and transaction volume) will not load the server the same way if one is executed using a 10Mbps connection and the other uses a 9600bps transfer rate. The server will be forced to reserve the data (process and threads, user-related data –whether fetched from a server or used to maintain the transaction open-, etc) for different periods depending on the speed of the link. When performance testing, one should thus take this aspect into account to attempt to create as realistic a spread of transaction speeds as possible.

Type of connection

Just as processors are limited resources, so are networks. Bandwidth is limited depending on the media used: copper, optical fiber, satellite link - all behave differently whenever a message is transmitted. This has an impact on the number of simultaneous communications and on the speed of each transaction. Contrary to common belief, communication does not occur at the speed of light: each media (copper, glass, space) has an impact on the speed of the packet of data. Distance becomes important when going via satellite links, as the message has to cover at least two times 36,000 kilometers (standard altitude for geosynchronous telecom satellites) generating a delay of about 240 milliseconds. To this transit delay, one must add all the other delays such as those to convert data from copper-based protocols to optical protocols, and from one packet switching network to another.

Another aspect one might need to take into account when simulating satellite links is the impact of solar flares and ionosphere on the quality of the data. These two aspects have an impact on the quality of the data and the availability of the satellites. Satellites become unavailable whenever solar storms send streams of high-energy particles, and any traffic must be routed by other types of connections. To perform realistic performance tests, such aspects need to be taken into account, instead of just simulating thousands of virtual users on a single 100MBps company LAN.

Turnaround time

TCP/IP slow start is a mechanism developed by the TCP protocol to attempt taking advantage of high- quality connections by sending larger and larger packets of data. Whenever an error is detected, the effective speed of transfer is divided by 2 and the packet in error is retransmitted. When a user accesses the server, this method is used, and if the last segment of the user's physical connection (the copper cable to the user's home) is subject to interferences, this will generate errors and negatively impact the speed at which the data is transferred. Thus the application server will have to keep in memory the user data for a longer period of time, effectively slowing down all other application users (by limiting the available resources).

Thus, when testing performances, the profile of the expected users (in effect their locations and types –and quality- of connections) have to be taken into account in order to be as realistic as possible.

Multiple network protocols

On a given network, there are a number of simultaneous protocols. Between the different servers of a multi-tiered application, there can simultaneously be over 30 different protocols. Each protocol is a process (in fact two: one at each end of the physical link), and as such has an impact on the CPU resources on the servers. But each protocol also sends –periodically – data on the link to keep the link alive. This has an impact on the availability of the limited link resource. Another solution implemented is that each protocol listens to the link and reacts only when something is being sent. This requires that the link be constantly monitored, and thus also has an impact on resources.

Some applications use multiple protocols to link users to servers, with each protocol dedicated to one type of data. This should also be simulated by performance test tools, but is seldom taken into account.

VPNs and encryption

To ensure the confidentiality of the data exchanged, and to allow remote users to access their application as if they were in the office, Virtual Private Networks have been developed, that physically encrypt data between the server and the user. Data encryption is a process that takes some time and resources both on the server and on the client's (user) side. Sometimes, in order to thwart eavesdroppers, data is constantly transmitted on the link to simulate exchanges between server and clients. This has an impact both on the server and user side (generating, sending, receiving and discarding data) and on the available bandwidth.

Security has become an increasingly important issue, with variable (time-dependent) keys being implemented, and performance testing tools have some difficulty to properly simulate this, especially when they limit themselves to recording data transferred and replaying it. In such cases the server discards the request and does not provide adequate responses. Due to the difficulty to recreate the adequate message (properly encrypted), performance testing does not address this issue and limits itself to not-encrypted transactions. This lack of realism is seldom evaluated, leading to defective performance analysis results.

Packet content

Whenever a block of data is exchanged between the server side and the user side, it becomes encapsulated according to the protocols used. The OSI model provides 7 different layers of protocols, each adding a set of data in front and behind the application data. Depending on the size of the data packets exchanged between server and user, the percentage of overhead vs. payload can be heavily weighted towards the overhead. This is of paramount importance when evaluating the type of link required. In some cases the author has seen a ration of 3 to 1 (i.e. 25% of the data exchanged was application data and 75% were network overhead). Thus if one expects to transfer 10000 bytes of data, the actual data on the network links can be 40000 bytes. And the effective rate of transfer of the application data should be divided by four.

Firewalls and security software

The large number of threats present on open networks has forced the use of firewalls and security software, to protect one's servers, network nodes and user PCs from malicious applications. These security measures constantly monitor the traffic and block any suspicious message. This has multiple impacts: it generates a process that uses resources that are in limited supply (CPU, memory, …), it requires periodic updates to keep up to date with the newest threats, and it may have a negative impact on outgoing transactions on the user's side, as they monitor all (incoming and outgoing) traffic.

Some security software requires the user to type a set of letters or numbers that are visually distorted to prevent malicious software from simulating real live users. Other security measures include sending data by other means (such as e-mails) to attempt to thwart unauthorized users. Testing the performances of such applications with automated tools requires monitoring multiple types of traffic (including e-mails), multiple applications (the application being tested and the e-mail service used), or using character recognition software to attempt deducing the valid data from the visually distorted information.

Security aspects are often bypassed or ignored when testing the performance of applications, leading to another layer of abstraction, another lack of realism.

Measurements

Often, time measurement are done on the server side, on the client (user) side and sometimes inbetween (with the use of dedicated agents located at different places on the network path between the servers and the clients). If one decides to measure the results at different locations on the server side, one should be able to evaluate the impact of the measurement on the aspect being measured. As in natural sciences, measurement can have an impact on what is being measured: the measurement tools should be not intrusive, but this is difficult when working on data passing through a network.

Agents are used to monitor some traffic using some protocols (not all traffic using all possible protocols). We have seen previously that multiple protocols can be used by a single application. We have also seen that network topology (i.e. the type of nodes and links) have an impact on the transactions and the speed of data transfer between server and users. We are aware that the impact of all these aspects (whether on the server, on the network or on the user) are multiple, if small, and that their sum may be significant. If the tools used for measurement are not of a sufficient level of precision, the data might be lost in the clutter. For example, if the measurement of transaction response time is in seconds, a delay of 240 milliseconds will not show up. Does that mean that it should be discarded ? Probably not, as it would add up to almost a second by the fourth transaction.

Evaluation of the impact

Each of the aspects covered in the two articles in this series impacts the response time and perceived performance from the user side. Taken by themselves each of these impacts is negligible, but their sum can be quite important. The importance will of course depend on the context, but should not be discarded out of hand.

With such a large number of aspects to take into account, simulating reality with performance testing becomes a challenge and a non-trivial task. Discarding some or all of the different aspects raised in the two articles is one way of attaining quicker (cheaper?) results, but at the expense of a lack of realism.

Before discarding any aspect, one should evaluate its impact with a thorough impact analysis. Discarding an aspect can then be done knowing the percentage of uncertainty added to the realism (or the increase in lack of realism). The sum of many small increases, not measured or considered negligible, often has a larger than expected impact on the results.

Conclusion

Performance testing is the one of the most challenging aspects of software testing in that it involves an extremely large number of variables, each impacting the results in one way or another. Removing some of these variables, either on the client side or on the server side, has a direct impact on the accuracy of the simulation.

Very often the criteria used in performance testing are not explicitly disclosed and almost never correctly evaluated in terms of the "percentage of uncertainty" associated with the data provided.

The large number of uncertainties in performance testing, whether they are on the client or on the server side, make it extremely challenging to predict accurately the behavior of applications when they are stressed. Removing possible causes for deviation and/or not taking into account the complexities results in simulations that are less and less representative of the way the application will behave in the field.

Testing the performances of applications involves non-trivial costs if one wishes to provide accurate data, and adequately "simulate reality". On the other hand, trying to provide quick (cheap?) results without taking into account the multiple aspects that can impact performances only results in creation of what is "really simulating" in the sense that it is a simulation (quite removed from reality).

Biography

Founder and principal consultant for TESSCO Technologies inc, Bernard Homès is a former member of the board of IEEE (France) and active in various software testing association and workgroups.
After 25+ years in software development and testing with different consultancies, he set up the TESSCO group consultancies and provides training and consultancy services worldwide.
The fields of operation of TESSCO Technologies inc. includes :

- Setting up Software Test Centres & Training testing teams,
- Specialization in mission critical systems (Banking, health care, telecoms, space & aeronautical systems),
- Qualification of Systems of Systems (large & complex critical systems) for international customers (Orange, Alcatel Alenia Space, Eurocopter).

Bernard Homès is also President of the French Software Testing Board (MailScanner hat einen möglichen Täuschungsversuch durch “www.cftl.net” festgestellt. www.cftl.fr) and represents France at the ISTQB.
A long time speaker at different international conferences and universities, Bernard has acquired a number of Software Testing certifications (incl. ISTQB Full Advanced certification), and chairs the ISTQB Advanced Level syllabus working party.

He can be contacted at bhomes@tesscogroup.com

RBCS

TIME TESTED. TESTING IMPROVED.

www.RBCS-US.com

Established in 1994, Rex Black Consulting Services, Inc. (RBCS) is an international hardware and software testing and quality assurance consultancy. RBCS provides top notch training and ISTQB test certification, IT outsourcing, and consulting for clients of all sizes, in a variety of industries.

RBCS delivers insight and confidence to companies, helping them get quality software and hardware products to market on time, with a measurable return on investment. RBCS is both a pioneer and leader in quality hardware and software testing - through ISTQB and other partners we strive to improve software testing practices and have built a team of some of the industry’s most recognized and published experts.

☎ : +1 (830) 438-4830 ✉ : info@rbcs-us.com

01. Consulting

- Planning, Testing and Quality Process Consulting
- Based on Critical Testing Processes
- Plans, Advice, and Assistance in Improvement
- Onsite Assessment

02. Outsourcing

- On-site Staff Augmentation
- Complete Test Team Sourcing
- Offshore Test Teams (Sri Lanka, Japan, Korea, etc.)
- USA Contact Person

03. Training

- E-learning Courses
- ISTQB and Other Test Training (Worldwide)
- Exclusive ISTQB Training Guides
- License Popular Training Materials

SCRUM & Testing: Back to the Future

by Erik van Veenendaal

Food for thought

One of the new methods currently being applied in many organizations is SCRUM, an agile method of project management that uses incremental work cycles known as sprints to provide frequent opportunities to assess and revise direction. As so often, however, it is simple but not easy! The SCRUM method doesn't say much about testing and in most books only unit testing and user acceptance testing are addressed. Have integration and system testing now become obsolete? Developers and business representatives should within agile development do the major part of the testing activities. It seems like utopia, but what does it look like in real-life? Exploratory testing is often identified as the main technique; what about our traditional test design techniques - are they no longer needed? These and many more questions need to be addressed when moving from a traditional development methodology to an agile methodology, such as SCRUM. When reading this column you may get the impression that I'm against SCRUM, agile development and the like. You could not be more wrong, but I would like to point out some testing issues from practice that one encounters when starting to apply SRCRUM. In other words; "food for thought" for the reader when moving from a traditional development methodology to agile development / SCRUM. By the way, my personal experiences are not from the games and internet industry, like many simplified stories one hears at conferences, but from more critical environments such as finance, embedded software and medical systems.

Test professionals

One needs to assess the current set of testers and decide whether they are "fit for agile".

Within agile testing, quality is the team's responsibility. The test analysts now have a different role. In addition to testing, they have to coach business representatives and developers on testing issues, review unit tests, etc. The tester is also involved in estimation sessions, defining the exit criteria (definition-of-done), reviewing stories and making them testable. All of this requires a senior person with good communication skills. We almost require ISTQB Advanced Level testers; persons that understand and are able to explain technical testing, business testing, test design and test management issues. What do we do with our junior testers? What do we do if we do not have senior testers with the right skills? Lisa Crispin in her book "Agile Testing" defines a number of qualities that agile testers should possess: deliver value to customers, have courage, respond to change, continuous feedback, etc. If you study those in more detail, there is little to no difference to what I would call a senior test analyst (in a traditional environment). If we analyse things, Lisa Crispin says that we need real professional senior testers with the right knowledge, skills and mindset. So what is new? Doesn't this sound familiar?

Developers

Within agile software development, unit testing is essential. Why is it essential only in agile, and why does it now get all the focus? It should have been essential in every lifecycle methodology! Can developers all of a sudden apply test design techniques? Most of them have never had any training in structured testing, not even at Foundation Level. Unit testing is not as easy as it seems, understanding test design is something not so common for most developers. Writing test code that makes the code fail as in XP does provide structural coverage, but not functional coverage. Getting developers into testing, needs a practical approach. We at Improve Quality Services often provide workshops with hands-on real-life practical cases to get them started. Test automation, essential with any incremental development methodology, is also needed, but as we as test professionals have already known for years, still proves to be a challenge for most organizations for various reasons. All we are saying is, we need high quality unit tests and fully automated regression tests. Things that have never been easy in the past. So what is new? Doesn't this sound familiar?

Test levels and test types

Most books on agile (testing) have a strong focus on unit testing and user acceptance testing. Whatever happened to good old integration and system testing? What about system integration testing in a systems-of-systems environment? Of course, there is not just one answer, since these terms mean different things in different organizations. The main problem I have with this approach is that again, after all these years, we have to re-explain to management why integration testing and system testing are needed. Caper Jones has already taught us that unit testing doesn't get far beyond a Defect Detection Percentage (DDP) of 35%

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

and together with only validation-oriented use cases as used in user acceptance testing, that just doesn't get the job done! Professional test cases using decision tables or the classification tree method are still required in many instances. Also, exploratory testing has its limitations (sorry, James). Some test levels or test types may take place outside a sprint, which seems impossible according to the agile theory, but this is what I see successfully happening in practice. Reliability and performance testing for example are often test types that run for more than four weeks and organizing an operational acceptance test or a system integration test within a sprint has been shown to be very difficult in practice. As a result, in practice some of the testing often takes place outside the SCRUM sprint. The main issue will then be to align the testing efforts and approaches. What needs to be done? Establish a test strategy that covers all testing activities (levels and types). To unambiguously define testing is usually a huge step in the right direction. This has been the issue in the past with the V-model and is also needed for agile software development. So what is new? Doesn't this sound familiar?

Many more challenges

The issues raised above are be no means the full list. I had to prioritize, keep it short and "write agile". Some other issues to be considered:

- What happens to the test manager, whose role is now obsolete?
- If we deliver software every four weeks, there will be stressful release periods more often. What does this mean for those involved?
- How do we organize agile testing with distributed teams, and how does it fit with outsourcing?
- Does management really understand the background to the agile manifesto? Even within SCRUM, there are limitations to change!
- How easy is it to get the right business representatives on-board and get them to do some testing? (In my experience, even reviewing test cases and providing input to a product risk analysis is often asking too much.)
- Etc.

The manifestos

Again, I would like to emphasize I'm not against SCRUM. It just isn't as easy as explained in the books and presented by some of the so-called gurus. If you are able to handle some or all of the issues, SCRUM projects have shown to be very successful. It usually helps enormously to have a detailed discussion about the agile manifesto with all those involved. What does it really mean? It is all about the philosophy behind the manifesto, understanding what is meant, and not about the actual wordings. Someone who really understands the agile manifesto, may even notice there is a lot of similarities with the test process improvement manifesto [Testing Experience, Issue 4, 2008]

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

Most of these statements can easily be adapted to agile software development. So what is really new? I "even" have a number of clients at TMMi levels 2 or 3, that practice SCRUM. In fact, those organizations that have the structured testing basics in place seem to be more successful at SCRUM than others. Again "food for thought"…….

I hope this column makes you aware of some of the testing issues that come with the implementation of SCRUM in an organization or project. Discuss them with team members inside and outside testing and management, and find practical result-driven solutions. Good luck in mastering SCRUM.

How Agile Methodologies Challenge Testing

by Rex Black

This article is excerpted from Chapter 12 of Rex Black's upcoming book Managing the Testing Process, 3e.

A number of our clients have adopted Scrum and other Agile methodologies. Every software development lifecycle model, from sequential to spiral to incremental to Agile, has testing implications. Some of these implications ease the testing process. We don't need to worry about these implications here.

Some of these testing implications challenge testing. In this case study, I discuss those challenges so that our client can understand the issues created by the Scrum methodology, and distinguish those from other types of testing issues that our client faces.

In my books and consulting, I typically recommend a blended testing strategy, consisting of three types of test strategies:

- Analytical risk-based testing;
- Automated regression testing;
- Reactive testing (also referred to as *dynamic testing*).

The blended testing strategy I recommend aligns well with Scrum and other Agile methodologies. In some cases, this strategy will mitigate the testing risks and reduce the testing challenges associated with these methodologies. However, it does not resolve all of the risks and challenges. In this article, let's examine some of the challenges that I've observed with RBCS clients using Scrum and Agile methodologies.

Dealing with the Volume and Speed of Change

One of the principles of Agile development is that project teams should "welcome changing requirements, even late in development" (see agilemanifesto.org). Many testing strategies, especially analytical requirements-based testing, become quite inefficient in such situations.

However, risk-based testing accommodates change, since we can always add risks, remove risks, change risks, and adjust the level of risk. If test execution is underway, we can adjust our plan for the remaining period based on this new view of quality risk. Since risk-based testing provides an intelligent way to decide what to test, how much, and in what order, we can always revise those decisions based on new information or direction from the project team.

Smart automated testing also accommodates on-going change. With care, automation at the graphical user interface can be maintainable. Automation at stable command-line interfaces does not tend to suffer severe maintainability problems.

The reactive testing that I recommend, not requiring much documentation, is also quite resilient in the face of change.

However, change can impose challenges for testing that are independent of the use of these test strategies. Many of these challenges arise from change in the definition of the product and its correct behavior (see also below). When the test team is not kept informed of these changes, or when the rate of change is very high, this can impose inefficiencies on the development, execution, and maintenance of tests.

Remaining Effective during Very Short Iterations

In sequential lifecycles, test teams can have a long period of time in which to develop and maintain their tests, in parallel with the development of the system, prior to the start of system test execution. Some more formal iterative lifecycle models, such as Rapid Application Development and the Rational Unified Process, often allow substantial periods of time between test execution periods for each iteration. These intervals allow test teams develop and maintain their test systems.

Agile methodologies like Scrum are less formal and faster moving. Consistent with the evocative name, *sprints*, these methodologies use short, fast-paced iterations. For a number of clients, RBCS consultants have seen this pace and brevity further squeeze the test team's ability to develop and maintain test systems, compounding the effects of change noted earlier. Testing strategies that include an automation element have proven particularly sensitive to this challenge.

The risk-based element of the recommended strategy can help. Risk-based testing focuses on the important areas of test coverage, and de-emphasizes or even cuts less important areas, relieving some of the pressure created by the short iterations. This ability to focus proves especially helpful for test teams also under tight resources constraints. Test teams in an Agile world should develop, maintain, and execute tests in risk priority order. Using risk priority to sequence development and maintenance efforts allows the test team to have the most important tests ready at the beginning of each sprint's test execution period.

Receiving Code after Inconsistent and Often Inadequate Unit Testing

Many of the authors, practitioners, and academics involved with Agile methodologies stress good, automated unit testing. Open-source test harness such as J-Unit and Cpp-Unit minimize the tool costs of doing so, overcoming one key obstacle to automation. Such automated unit

tests allow what Agile proponents call *refactoring*. Refactoring is the redesign of major chunks of code or even entire objects. The automated unit tests provide for quick regression testing of refactored code. Some Agilists recommend substituting automated unit tests for design, as in Test Driven Development. While this sounds good in theory, there are two problems that we tend to observe with this approach in practice.

First, unit testing has limited bug-finding utility. Capers Jones has found 25 to 30% average defect removal effectiveness for unit testing.[1] RBCS assessments have shown that good system testing by an independent test team averages around 85% defect detection effectiveness. So, while unit testing helps, the main filter to prevent excessive field failures remains system testing.

Second, we find that, under the guise of excuses both valid and not-so-valid, many programmers do not create automated unit tests and in some cases don't do any unit testing at all. This creates a great deal of trouble for the system test team, which, as mentioned above, remains a critical bug-removing filter. The short test execution periods on Agile sprints, compared to sequential projects, means that the degree of damage caused by one or two days' of test progress blockage due to highly buggy code is higher than in a sequential project.

Delivery of unstable, buggy code will undermine one of the key benefits of the risk-based testing portion of the recommended strategy, which is the discovery of the most important defects early in the test execution period. It also inevitably leads to a large degree of code churn during testing, since so much must change to fix the bugs. The amount of change can ultimately outstrip even the ability of the best automated regression test system to keep up, which would then lead to lower defect detection effectiveness for the test team.

Managing the Increased Regression Risk

Capers Jones has found that regression accounts for about 7% of bugs.[2] In iterative lifecycles like Scrum, though, code that worked in previous sprints gets churned by new features in each subsequent sprint. This increases the risk of regression. Agile methodology advocates emphasize good automated unit testing in part to manage the regression risk inherent in such churn.

However, good unit testing has limited defect removal effectiveness, as cited earlier. So, automated regression testing via unit tests will likely miss most of the regression bugs. Therefore, we need effective regression testing at the system test level (which has a higher level of defect detection effectiveness). By combining risk-based testing with the automated regression testing, test teams can effectively manage the increased regression risk.

Making Do with Poor, Changing, and Missing Test Oracles

Agile methodologies de-value written documentation. Special scorn is reserved for specifications. For example, the Agile Manifesto suggests people should value "working software over comprehensive documentation." This creates real challenges for a test team. Testers use requirements specifications and other documents as *test oracles*; i.e., as the means to determine correct behavior under a given test condition. We have seen testers in Agile situations given documents with insufficient detail, or, in some cases, given no such documents at all.

Even the project team provides the test team with adequate documents, two other Agile development principles keep the test oracle challenge alive. First, Agile requires teams to embrace change, as I discussed earlier. Second, Agile principles state that "the most efficient and effective method of conveying information to and within a development team is face-to-face conversation" (see agilemanifesto.org). For many of our clients following Agile methodologies like Scrum, these two principles allow the project team to change the definition of correct behavior at any time, including after testers have tested to confirm a particular behavior and after testers have reported bugs against a particular behavior. Further, the definition of correct behavior can change in a meeting or a discussion which might not involve the test team and which might produce no documented record of the change.

No known test strategy can resolve this challenge. Resolving the challenge requires change management. The percentage of rejected bug reports provides a measurable symptom of this challenge. Rejected bug reports are ones that the team ultimately discarded because the reports described correct behavior. Projects with well-defined, stable test oracles enjoy bug reject rates below five percent. Projects with poor, changing, or missing test oracles often endure bug reject rates of 30 percent or more.

We have estimated imposed test team inefficiencies from such test oracle problems at around 20 to 30 percent. Further, the inability to determine precisely whether a test failed affects both the efficiency of the testing and the defect detection effectiveness. When testers spend time isolating situations that the project team ultimately chooses to define as correct behavior, that takes away time they could have spent finding and reporting real bugs. These bugs create subsequent problems customers, users, and technical support staff, and distractions for developers and test teams.

Further, the situation creates frustration for the testers that reduces their morale and, consequently, their effectiveness. Testers want to produce valid information. When much of the information they produce – in the form of rejected bugs reports – ends up in the figurative wastebasket of the project, that tends to make people wonder why they bother.

It's important to realize that this reduction in test effectiveness, efficiency, and morale is a potential side-effect of Agile methodologies. Organizations using Agile methodologies must assign responsibility for these outcomes in the proper place. Bad problems can get much worse when the test team is held accountable for outcomes beyond their control.

Dealing with a Shifting Test Basis

Requirements-based testing strategies cannot handle vague or missing requirements specifications. Missing requirements specifications would mean a test team following a requirements-based testing strategy not only can't say what it means for a particular test to pass or fail, they wouldn't have a *test basis*. The test basis is that which the tests are based upon. Requirements-based testing strategies require test teams to develop test cases by first analyzing test conditions in the requirements, and then using those test conditions to design and implement test cases. A missing or poor requirements specification won't work for such analysis.

The test basis also provides a means to measure the results. In a requirements-based test strategy, testers can't report test results accurately if the requirements are missing or poor. Testers can't report the percentage of the test basis covered by passed tests, because the requirements won't provide enough detail for meaningful coverage analysis.

With the risk-based testing strategy recommended in my book, test teams can evade both problems. For the test basis, testers use the quality risk items. They design and implement test cases based on the quality risk items. The level of risk associated with each risk item determines the number of test cases and the priority of the test cases derived from that risk item. The test team can report test results in terms of quality risks mitigated versus not mitigated.

From Detailed Documentation to Many Meetings

As the Agile Manifesto quotation cited earlier puts it, people should focus on creating working software rather than comprehensive documentation. However, the information that was, under the *ancien regime*, captured and exchanged in these documents must flow somehow, so Agile advocates promote "face-to-face conversations." This, of course, is another name for a meeting. From a marketing perspective, it was smart of the Agile advocates not to use the word *meeting* in the Agile Manifesto, but the reality remains.

I mentioned in an earlier section on test oracle issues with Agile methodologies the problem of a meeting or a discussion that changes the definition of correct behavior, which did not involve the test team, and which did not produce a documented record of the change. The flip side of this problem, in some organizations, is that everyone gets invited to every

1 See Jones' article "Measuring Defect Potentials and Defect Removal Efficiency," found in the Basic Library at www.rbcs-us.com.

2 See, for example, Jones' figures in Estimating Software Costs, 2e.

meeting, the meetings balloon in number and duration, managers and leads are not available to manage and lead their team because they are in meetings much of the day, and effectiveness and efficiency drop.

One manager jokingly described this situation as follows: "Scrum is a heavyweight process. I'm surprised at the name *Agile* – it should be called *couch potato*. There are too many meetings. There's too much jawboning. I find it really ironic that there are all these books explaining how simple it is."

To be fair, having too many meetings is a problem that any project following any lifecycle model can suffer from. I've worked on classic waterfall projects as a test manager where I spent four hours or more per day in meetings. I had one client relate a hilarious anecdote where a senior manager, in response to a complaint from a line manager about how attending meetings was negatively impacting his ability to lead his team, shouted, "We're going to continue to have these meetings until I find out why nothing is getting done around here!"

That said, every organization, every project, and every lifecycle has to strike the right balance. In some cases, organizations and projects following Agile methodologies react too strongly to the Agile Manifesto's comments on documentation and face-to-face "discussions." Further, embracing change should not meet throwing out or re-considering previous decisions to the extent that it paralyzes the team. Teams using Agile methodologies must achieve the right balance between documentation and meetings. Teams using Agile methodologies must have crisp, efficient meetings that move the project forward and course-correct, not meetings that grind the team down and go around in circles.

Holding to Arbitrary Sprint Durations

Some of our clients following Agile methodologies like Scrum tend to ritualize some of the rules of the process. The time deadlines for sprints seem particularly subject to this ritualization. At first, I was puzzled that these same clients discarded some other rules, such as the requirement for unit testing, often with less reason than the reasons for which they held tightly to the deadlines. However, deadlines are tangible, while the benefits of unit tests are not as well-understood and clear, so I can now see the reasons behind the selective emphasis on certain Agile process rules over others.

For example, suppose that a project team follows four week sprints. A systemic problem in our industry relates to software estimation, particularly the tendency to over-commit in terms of the number of features (*user stories*) for that sprint. So, on the last Friday of the sprint, with development ending late for the sprint, the arbitrary deadline remains intact at the expense of the test team's weekend.

Fully resolving this challenge requires team and management maturity. When people habitually and systematically over-commit, management should require use of historical progress metrics for estimation. Many Agile practitioners use metrics like *burndown charts* and *story-point velocity*. The Scrum process includes gathering and using such metrics.

If fixing the software estimation problems cannot occur, risk-based testing helps the test team deal with systematic and constant over-commitment. To start with, when the test team is time-crunched over and over again at sprint's end, the test team should accept that the project team's priorities are schedule-driven, not quality-driven. The test team should revise the risk analysis approach to institute an across-the-board reduction in the extent of testing assigned to each quality risk items during risk analysis for subsequent projects. That way, at least the test team won't over-commit.

In some cases, in spite of reducing the scope of testing, the test team still can't execute all the tests in the available time at the end of a sprint. If so, rather than ruining their weekend to run every test, the test team can select the most important tests using the risk priority number. Less important tests can slip into the next sprint. (You'll notice that Scrum and other Agile methodologies allow user stories to slip from one sprint to the next in the same way.) Of course, if the test team must consistently triage its tests in this fashion, they should again adjust the test extent mapping downward for future sprints.

Dealing with Blind Spots in the Sprint Silos

Not all experts on Agile methodologies agree on the need for independent test teams. Some seem to think of testing as a subset of the tasks carried out by programmers on an Agile team, specifically creating various forms of automated unit tests and/or acceptance tests. This might exclude an independent test team, or it might transform the role of that team.[3]

Most of our clients adopting Agile methodologies have retained the independent test team, or at least the independent tester, to some extent. For those retaining the independent team, most have chosen to partition the team across the sprints in some way, often making each tester answerable on a day-to-day task basis to the Scrummaster or other sprint leader, rather than to the test manager. In the case of having independent testers but not independent test team, there is no test manager.

This provides some advantages, especially to the person leading the sprint:

- Each tester focuses entirely on sprint-related tasks, with minimal outside distractions.
- Each tester allocates time entirely to the benefit of the sprint and its exigent goals.
- The sprint leader can re-direct any tester to focus on what is most important to the sprint team, often without having to consult the test manager.
- The sprint leader can – and, at the end of a sprint, often will – call on the tester to put in extra efforts to hit sprint targets.
- The amount of test effort allocated to the sprint does not vary once the number of testers for the sprint is determined, and the test manager cannot surprise the sprint leader with a sudden reduction in test effort to serve other test team priorities.

As you can see, some of the advantages have zero-sum-game elements that carry within them the seeds – if not the actual fruit – of various potential problems.

The challenges of this approach – some of which are obvious corollaries of the advantages – include the following:

- The tester thinks of himself as – and conducts himself as – less of a tester and more of a specialized developer. This can include a loss of interest in growing test-specific skills in favor of growing technical skills.
- The tester has less contact with people outside the sprint team, reducing the wider, system-level perspective provided by an independent test team.
- In the absence of coordinative guidance from the test manager, the tester starts to make mistakes related to gaps and overlaps. The tester fails to perform some test tasks that make sense, especially longer-term investments that don't necessarily benefit the current sprint. The tester redundantly performs test tasks done by other testers on other sprints, because she wasn't aware of the other testers' work.
- The test manager, having lost the ability to manage the workload of their resources, finds the morale suffers and turnover increases as unsustainable workloads at the end of each sprint take their tool.
- The ability of the test team to grow a consistent, powerful, maintainable test system – the test scripts, the test tools, the test data, and other stuff needed for effective and efficient testing in the long-term – goes down because of the exigent focus on the sprint's immediate needs.
- Testers display a tendency to "go native" and lose some of the objectivity that an independent test team normally provides, suffering a loss of defect detection effectiveness.

None of these challenges (or advantages, for that matter) arises from Agile methodologies

3 See, for example, the abstract of Kent Beck's talk, given at the Quality Week 2001, conference, www.soft.com/QualWeek/QW2001/papers/2Q.html. Beck claimed that unit testing by programmers might eventually make the defect detection role of independent test teams (referred to as "QA" in the abstract) superfluous, transforming testing into a role similar to a business analyst's. My earlier comments about unit testing, and the figures I cited from Caper Jones' work on the limits of the effectiveness of unit testing, make me skeptical that we will see substantially bug-free code delivered to test teams, no matter what the lifecycle model, but I'd enjoy working on projects where someone managed to prove me wrong.

Foto: © Heng Kong Chen - Fotolia.com

INTERNATIONAL SOFTWARE QUALITY INSTITUTE

CERTiFYING PEOPLE

SECURITY
PROJECT MANAGEMENT
REQUIREMENTS ENGINEERING
INNOVATION MANAGEMENT
QUALITY ASSURANCE
SOFTWARE ARCHITECTURE
CONFIGURATION MANAGEMENT
SOFTWARE TEST
SPICE
RFID

Meet us at CONQUEST
16-18 September 2009
Nuremberg, Germany

conquest-conference-.org

www.isqi.org

iSQI

per se; it just happens that Agile methodologies as typically practiced tend to accentuate them. I have observed similar problems for years on projects following sequential models or no model at all, when the test team adopted what I refer to as the *project resource* approach to test team organization.

The challenges are not insurmountable, if an independent test team exists. The test team, lead by a good test manager, can introduce centripetal forces that will bind the team together and makes its actions consistent and congruent. These forces then balance the sprint-specific centrifugal forces that tend to push the test team members into sub-optimizing for their sprint as well as isolating the test team members from broader testing considerations.[4]

One approach to managing these challenges is to have a separate test period that follows the development sprint. (To keep the terminology clean, some like to call this approach an alternation of *development sprints* following by *test sprints*.) Some of our clients have pursued this approach. It does seem to work for them, when the test team remains engaged in the development sprint. In other words, if the test team disengages from the development team during the development sprint, you are just exchanging one form of siloing for another.

Managing Expectations

To close this article, let's look at a psychological challenge to test teams on Agile projects. Gartner, the industry analysts, say that IT industry adoption of new technologies and ideas goes through a *Hype Cycle* (see www.gartner.com/pages/story.php.id.8795.s.8.jsp). The Hype Cycle consists of five distinct phases, as they describe on their Web site:

1. **Technology Trigger**. The first phase of a Hype Cycle is the "technology trigger" or breakthrough, product launch or other event that generates significant press and interest.
2. **Peak of Inflated Expectations**. In the next phase, a frenzy of publicity typically generates over-enthusiasm and unrealistic expectations. There may be some successful applications of a technology, but there are typically more failures.
3. **Trough of Disillusionment**. Technologies enter the "trough of disillusionment" because they fail to meet expectations and quickly become unfashionable. Consequently, the press usually abandons the topic and the technology.
4. **Slope of Enlightenment**. Although the press may have stopped covering the technology, some businesses continue through the "slope of enlightenment" and experiment to understand the benefits and practical application of the technology.
5. **Plateau of Productivity**. A technology reaches the "plateau of productivity" as the benefits of it become widely demonstrated and accepted. The technology becomes increasingly stable and evolves in second and third generations. The final height of the plateau varies according to whether the technology is broadly applicable or benefits only a niche market.

As we approach the 2010s decade, we are seeing Agile methodologies in the peak of inflated expectations phase. RBCS clients have inflated expectations for Agile methods that relate to quality, productivity, and flexibility.

Some test teams on Scrum projects report to management that the product quality is no higher than normal, and sometimes even lower than normal. Some test teams on Scrum projects report to management that the challenges discussed in this appendix have reduced their efficiency. Some test teams on Scrum projects report to management that, while development might not experience negative consequences or pain from change – which is a key promise of Agile methodologies and one of the key management selling points – testing will still endure problems when coping with unlimited, unmanaged change.

When these test teams on Scrum projects report these negative outcomes to management, management experiences a psychological phenomenon called *cognitive dissonance*. Cognitive dissonance involves feelings of mental tension between the incompatibility of their expectations of Agile methodologies and the observed results. Ultimately, these cognitive dissonance experiences, across the various adopters of Agile methodologies, will push these approaches along the Hype Cycle, out of the peak of inflated expectations. In the short run, though, management might engage in another psychological phenomenon called *projection*. This involves projecting onto others how you feel about something they are associated with, but perhaps not responsible for. The experienced test professional knows this phenomenon better under the phrase *killing the messenger*.

Conclusion

The test strategies I typically recommend will support the stated goals of Agile methodologies. Risk-based testing supports increased quality, since it focuses testing on high-risk areas where testing can significantly reduce the risk. Risk-based testing supports increased productivity, since it reduces or eliminates testing where the quality risk is lower. Risk-based testing supports flexibility, since it allows regular revision of the quality risk items which re-aligns the remaining testing with the new risks and their new levels of risk. Automated regression testing helps to contain the regression risks associated with Agile methodologies, allowing a higher rate of change. Reactive testing allows testers to explore various aspects of the system that risk-based testing and automated regression testing together might miss.

However, this blended risk-based, automated, and reactive test strategy cannot fully resolve the challenges covered in this case study. In the long run, people will come to recognize that, and rational trade-offs will prevail. In the short run, though, while Agile methodologies remain on the peak of inflated expectations, the test team must be careful to communicate any testing issues that arise due to Agile methodologies and their effect on testing rather than from testing itself.

4 See Chapter 8 of my book Managing the Testing Process. The first edition (1999), the second edition (2003), and the upcoming third edition (2009) all make the same point. Plus ca change, plus c'est la même chose...

Biography

With a quarter-century of software and systems engineering experience, Rex Black is President of RBCS (www.rbcs-us.com), a leader in software, hardware, and systems testing. For over a dozen years, RBCS has delivered services in consulting, outsourcing and training for software and hardware testing. Employing the industry's most experienced and recognized consultants, RBCS conducts product testing, builds and improves testing groups and hires testing staff for hundreds of clients worldwide. Ranging from Fortune 20 companies to start-ups, RBCS clients save time and money through improved product development, decreased tech support calls, improved corporate reputation and more. As the leader of RBCS, Rex is the most prolific author practicing in the field of software testing today. His popular first book, Managing the Testing Process, has sold over 35,000 copies around the world. His five other books on testing have also sold tens of thousands of copies. He has written over twenty-five articles, presented hundreds of papers, workshops, and seminars, and given about thirty keynote speeches at conferences and events around the world. Rex is the President of the International Software Testing Qualifications Board and a Director of the American Software Testing Qualifications Board.

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)

2 years subscription

☐ **60,- €**
(plus VAT)

Date

Signature, Company Stamp

© iStockphoto.com/Zemdega

A Community

over 110,000 C

Are you on th

www.is

*March 2009


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


© iStockphoto.com/mbbirdy

Testing in an organisation going agile

by Eric Jimmink

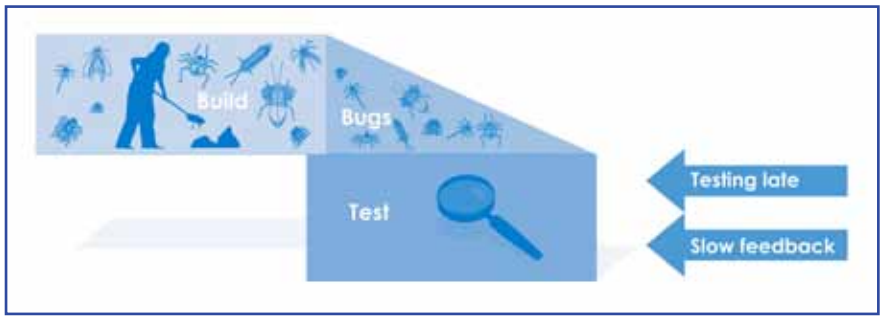
One of the biggest misconceptions about organisations adopting agile development, is that testing as a whole (and testers as individuals) can adapt to the change without much help, or without any changes in the staff. Is it realistic and fair to expect the same people to do something fundamentally different, and then to have them do well?

In this article, I will share some insights that may help organisations make the changes in testing successful and sustainable.

First of all, any organisation should make a clear policy statement regarding team composition and stability. E.g., *all members of agile teams (including testers) shall stay with their team for the duration of a project from start to finish, or longer. As a rule of thumb, no-one shall be placed in more than one team.*

Secondly, all testers in the organisation should possess the skills, mindset and background knowledge required to operate well in an agile environment. *Staff training and recruitment selects and instils 'Agile DNA' in all testers.*

Lastly, the organisation should acknowledge testing roles and styles. E.g., *our company identifies technical system testing and domain/business level testing as different roles on a project team, and as career paths for testing professionals.*



Picture 1

Picture 2

Pictures (1&2): The effects of late and early testing

Involving testers from start to finish

At first glance, it may look wasteful to involve testers before there is any code. However, we all know that early testing can save lots of time. Agile teams can take this to extremes. Besides testing code immediately after it is integrated, or helping developers write code test-first, testers can and should talk to customers and help improve requirements long before they enter the iteration. Rarely will a requirements specification or design ever be flawless the first time around. That is fine once a team gets used to emergent designs, and requirements that are not frozen before coding (or even testing) can commence. As such, the added value of testing is undeniable.

Testing prior to the project

How early can you start testing? Personally, I have been asked to participate as a tester in project feasibility studies long before the actual teams were formed. I have reviewed and even formally inspected project proposals, checking them against the RFP. Most defects found at that stage are critical for determining the project's success and profitability.

No-one shall be placed in more than one team

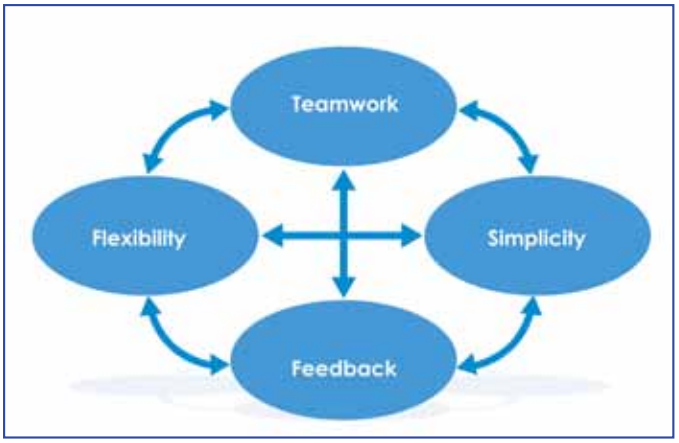
The latter part of the policy statement is easily clarified: dividing attention across multiple teams seriously detracts from one's focus and effectiveness. For the business, this is an important point. It may require investments such as training and task rotation. Some of your people have to be able to wear different hats and do work outside their own field of expertise. That's less wasteful than dividing work between teams, and more fun to do as well.

For example, say that you expect that most teams will need about 1.5 people for testing. You can't select 1.5 testers, so you must choose either 1 or 2. Some teams would have 2 testers, and many teams would have to cope with just one. As a tester, you will often have to delegate test execution to a colleague who does not have a background in testing. Keep your test cases and designs simple enough for your other team members to understand.

Picture 3: The four values in agile testing

Testers with 'Agile DNA'

Training can and should be given in the basics



Picture 3

and the underlying principles of the agile way of working, to ensure that it is properly understood and executed. One should also monitor those things, and coach where needed. If I see team members who are seated within 3 metres sending emails to each other, or logging all defects into a bug tracker, I feel compelled to ask them Why. If a developer is reluctant to show me his unit test, I will try to convince him. He should take pride in showing me his unit test and appreciate my feedback; we're part of the same team now, right? Besides, if I can see the unit tests, I can avoid producing tests that overlap.

Part of the Agile DNA is the mindset and attitude that can be learned on the job. As a tester, your primary function is to facilitate the team by giving fast feedback. This stresses the value of test *execution* over test design. Often you will have to be flexible and change testing tasks at a moment's notice. For example, during the daily scrum you will hear what has changed in the project, and which feedback is wanted the most. Doing that which adds the most value to the team at any given point in time, that's what agile working is all about. Being pro-active adds more value than reactive behaviour. Don't just signal a defect in a bug tracking system, when you can readily help to fix the problem. Many defects can be fixed right away if you communicate (demonstrate) what is wrong. If you spot an oversight or ambiguity in the requirements, then speak up (literally) and resolve the matter directly with your product owner.

Recruitment of agile testers

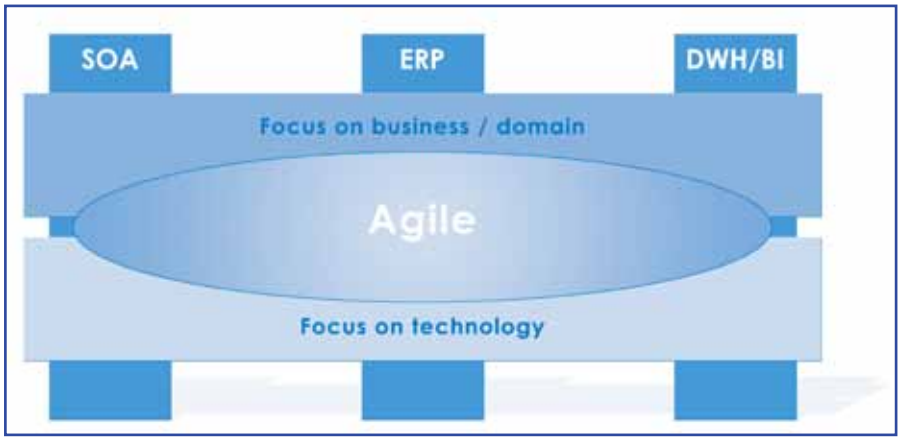
Most soft skills are hard to train, and many are in fact vital for doing well in an agile environment. In the past, recruitment in many companies was selecting candidates on their technical merit and their analytical skills, and pushing the less communicative ones into the testing field. That seems fine when you use a waterfall-style development model with neat procedures and documents. Testers being at the end of the chain, their need to communicate is not so great, right? How different is that when going agile. Testers can and should become the hub in the web, communicating with all other disciplines.

Recruitment of agile testers should therefore select candidates that communicate well. It would be much easier for a team player with good soft skills to learn technical testing skills and knowledge, than it is for a nerd with lots of testing experience to learn how to communicate effectively. In times of reorganisations, this provides a new chance. Recruitment can also be done within the company. People who have become obsolete in their own line of work, may possess the skills and mindset to become good agile testers.

Acknowledging testing roles and styles

An agile team needs both technical testers and business oriented testers.

Why this division into two kinds of testers? Because they require different skills and knowledge. They are becoming specializations, with most people having aptitude to excel in only one of the two. (There is a third category of testers which is less desirable for agile, although they form a staple resource for waterfall-style projects. Sadly, many testers belong to that category. An agile company needs to educate the promising individuals, and lose the others.)

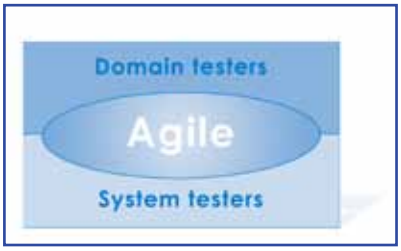


Picture 4

Picture 4: Defining testing at two levels

At EuroStar in 2008, I noted that there were unifying factors in most projects. I took three popular forms of business process modelling, and looked at the testing in those situations from both a business and a technical perspective. For example, in a SOA project, you have a top level where services are orchestrated into business processes. In testing, you want to assume that the individual services have already been tested thoroughly. Your test cases will be formulated in terms of the business, and in order to do so you will communicate a lot with people from the business - end users and domain experts. Conversely, the testing of the component services will be much more technical in nature, relying on tools, technical skills, and knowledge of the implementation platform. A tester at that level will communicate much more with the developers, and a bit less with the business.

Another unifying factor of all application areas was that business problems tend to be complex in nature. This means that incremental delivery and control is desirable, and that the projects should accommodate this. In short, the projects should be managed in an agile fashion.



Picture 5

Picture 5: Summarized: Ordina's vision on testing

It is a shared vision within my company, that we shall do more and more projects using agile methods. And that we need two kinds of testers to do so effectively. We are already undergoing changes and investing in our testers to make them more agile. It's not a small step, to say that you are defining two kinds of specializations for testing professionals. The choices have to be made meaningful by providing career paths, and specific training. That training may encompass domain-specific knowledge for defined pieces of the market, and detailed knowledge of tools and platforms. All such knowledge is somewhat transient in nature, but is nonetheless needed and must be maintained to remain truly responsive and agile as an organisation.

Conclusion

When an organisation chooses to embrace agile methods, testing must undergo a lot of changes to fit in and add the most value. Testers must possess the proper mindset and expertise. The organisation can help a lot by providing stable teams and by matching training, career paths, and recruitment to the new situation. One of the keys to success lies in acknowledging two levels of testing that may require two different kinds of testers.

Biography

Eric is a test consultant at Ordina.
He started his career as a software developer / lead engineer. Eric has been involved in agile projects since 2001. As a person, he is a firm believer in learning by doing, leading by example, and thinking outside of the box.
Last year, Eric spoke at EuroStar and at Agile2008. He is co-author of '*Testen2.0 – de praktijk van agile testen*', a Dutch book about testing in the context of agile development.
At the Agile Testing Days in Berlin, Eric will cover a vastly different topic: 'Promoting the use of a quality standard'. Equally, that involves the team and the whole organisation, not just the testers.


© Katrin Schülke

k a n z l e i

h i l t e r s c h e i d

Berlin, Germany

IT Law
Contract Law

German
English
French
Spanish

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

© iStockphoto.com/ausinasia

Mistake-Proofing

by Mary & Tom Poppendieck

This article is an excerpt from the forthcoming book: Leading Lean Software Development, By Mary and Tom Poppendieck, published by Addison Wesley, ISBN 0321620704.

All software development methodologies have the same objective: Lower the cost of correcting mistakes by discovering them as soon as they are made and correcting them immediately. However, various methodologies differ dramatically in the way they frame this goal. The sequential life cycle frame is to create a complete, detailed design so as to find defects by inspection before any coding happens – because fixing design defects before coding is supposedly less expensive than fixing design defects later on. This theory has a fundamental problem: It separates design from implementation. As we discussed in Chapter 1, separating decision-making from implementation is one of the biggest causes of waste in product development. Good designs come from hands-on engineering, lots of prototypes, and frequent feedback cycles. The sequential development frame – as typically implemented – has no place for go-to-the-source feedback.

But it gets worse. The sequential frame leaves testing until the end of the development process –exactly the wrong time to find defects. Every development frame we know of says that finding and fixing defects late in the development cycle is very expensive – including the sequential frame. And yet, sequential frames have brought about exactly the result they are trying to prevent – they delay testing until long after defects are injected into the code. In fact, people working in sequential frames ***expect*** to discover defects during final testing – often a third or more of a software release cycle is reserved to find and fix defects after coding is theoretically "complete." The irony is that the sequential frames are designed to find and fix defects early, but they invariably end up finding and fixing defects much later than necessary.

Where did we get the idea that finding and fixing defects at the end of the development process is appropriate? Just about every quality program in existence teaches exactly the opposite. The rational approach is to mistake-proof the development process so that the moment a defect in injected into the code, it will be detected immediately and fixed on the spot. We know that it is infinitely easier to find defects that have just been inserted into defect-free code than it is to find defects buried layers deep in defect-ridden code. And yet we save testing until the end of the development process. Why do we do this?

Part of the reason why we delay testing is because testing has historically been labor intensive and difficult to perform on an incomplete system. In lean development, we no longer accept this historical frame – we change it. In fact, changing the historical mindset regarding testing is probably the most important characteristic we have seen in successful agile development initiatives. You can put all of the agile project management practices you want in place, but if you ignore the disciplines of test-driven development and continuous integration, your chances of success are greatly diminished. The so-called "engineering disciplines" ***are not optional***, and should not be left to chance.

Test-Driven Development

The challenge laid down by Edsger Dijkstra in 1972 was: "Those who want really reliable software will discover that they must find means of avoiding the majority of bugs to start with, and as a result the programming process will become cheaper. If you want more effective programmers, you will discover that they should not waste their time debugging – they should not introduce bugs to start with." [1]

So how do you keep from introducing defects in the first place? Dijkstra was clear on that; you have to construct programs "from the point of view of provability." This means that you start by stating the conditions which must hold if the program is to perform correctly (in other words, start with the tests which must pass). Then you write the code which makes the conditions come true (makes the tests pass).[2] Dijkstra called this structured programming, but somehow in all of the years of structured programming, his clear, simple concept got lost. Then in the late 1990's, Kent Beck, Erich Gamma and others revived Dijkstra's concept and called it Test-Driven Development (TDD). Beck and Gamma created a tool (J-Unit) which has been widely used and adapted to just about every language out there. In 2002 Ward Cunningham expanded the concept of TDD to include functional tests with FIT (Framework for Integrated Testing), a tool which has been extended by many others.[3]

Continuous Integration

Code does not exist in a vacuum – if it did, software development would be trivial. Code exists in a social setting, and arguably the biggest problem we have in software development is that code isn't always a good neighbor. Thus integration is often the biggest problem with software, and when it is, leaving integration to the end of development is another example of a defect-injection process. Harlan Mills figured this out 1970 – and he found a solution. First he developed a skeleton of a system, and then he added small programs one at a time and made sure they each worked before adding the next program. He called this top-down programming, but somehow, the underlying idea got lost in translation and few people really understood what Mills meant by the term.

Mills' explained it this way: "My principle criterion for judging whether top down programming was actually used is [the] absence of any difficulty at integration. The proof of the pudding is in the eating!" [4] Today our term for what Mills called top-down programming is "continuous integration," and we use the same criteria for judging its effectiveness: If you have a big bang integration problem – or

1 (Dijkstra, The Humble Programmer, 1972)

2 (Freeman, 1975) p 489.

3 See (Mugridge & Cunningham, 2005)

4 (Mills H. , 1988) p5.

even a little bang integration problem – you are not effectively doing continuous integration.

Where did we get the idea that we should wait until the end of development to put the pieces of the system together? This is just another example of saving up all of our hidden defects and not even trying to find them until the end of development. This violates every known good testing practice in the book – how can we possibly think it makes sense? Why do we do this?

The explanation lies partly in the distorted logic of sequential frame, but mainly in the sheer difficulty of testing software after every tiny integration step. First of all, testing has been laborious, and secondly, testing partially done code has an annoying tendency to raise false negatives. In addition, as we raise the level of integration to larger system environments, the testing becomes all the more complex, because we have to test across a much wider span of programs. And finally, it is impossible, really, to find everything that could go wrong, and testing for only a few of the possibilities can be a very long process, and therefore one that cannot be undertaken very frequently.

So it should come as no surprise that continuous integration is not easy, and as integration escalates up to higher system levels, it becomes a huge challenge. But consider the benefits: Many companies set aside 30% or more of each release cycle for a series of integration tests and the associated repairs necessary when the tests uncover defects. On the other hand, the best companies budget no more than 10% of a release cycle for final verification, and they do this with increasingly short release cycles. They can do this because the defects have been uncovered earlier in the development process; if they find defects at final verification, they are surprised and work to find out how to keep that defect from escaping to the final verification step in the future.

Bibliography

Dijkstra, E. W. (1972). The Humble Programmer. *Communications of the ACM , 15* (10), 859-866.

Freeman, P. (1975). *Software Systems Principles: A Survey*. Science Reserach Aoosciates.

Mills, H. (1988). *Software Productivity*. Dorset House.

Mugridge, R., & Cunningham, W. (2005). *Fit for Developing Software: Framework for Integrated Tests*. Prentice Hall.

Subscribe at

testing experience

www.testingexperience.com

Biography

Mary Poppendieck has been in the Information Technology industry for over thirty years. She has managed software development, supply chain management, manufacturing operations, and new product development. She spearheaded the implementation of a Just-in-Time system in a 3M video tape manufacturing plant and led new product development teams, commercializing products ranging from digital controllers to 3M Light Fiber™.
Mary is a popular writer and speaker, and co-author of the book Lean Software Development, which was awarded the Software Development Productivity Award in 2004. A sequel, Implementing Lean Software Development, was published in 2006. A third book, Leading Lean Software Development, will be published in late 2009.

Tom Poppendieck has 25 years of experience in computing including eight years of work with object technology. His modelling and mentoring skills are rooted in his experience as a physics professor. His early work was in IT infrastructure, product development, and manufacturing support, and evolved to consulting project assignments in healthcare, logistics, mortgage banking, and travel services.
Tom holds a PhD in Physics and has taught physics for ten years. He is co-author of the book Lean Software Development, which was awarded the Software Development Productivity Award in 2004. A sequel, Implementing Lean Software Development, was published in 2006. A third book, Leading Lean Software Development, will be published in late 2009.

The Future of Testing - How Testing and Technology will Change

by Joachim Herschmann

This article is based on a talk I have been giving over the last year or so at various conferences and seminars in some form or other. During these challenging economic times I have seen a dramatic increase in the desire of QA professionals to understand better where the software testing industry as a whole is heading and how testing processes and the technology involved will most likely change. There is no doubt that in coming years test and quality professionals and development organizations as a whole will be under ever-increasing pressure to test better and test faster. In the face of rapid technology changes, increasingly complex systems, globalization, and new regulatory burdens, changes in the way we test - and develop - software are critical. Deeply embedded software, SOA, mobile devices, cloud computing, and technologies of which we aren't even aware affect testing and often make it more difficult. Dynamically interacting systems that continuously morph themselves make it almost impossible to know what we are even testing. While new approaches like agile and lean development are rapidly gaining popularity, they pose new challenges as well as opportunities for software quality.

While certainly no one can predict the future, I have at least some ideas about what I believe is the direction the software industry and testing is going in. I will share with you some of these ideas and describe the emerging trends that will impact testing. Let's have a look at how these powerful forces will result in burgeoning new test technologies and require software leaders to change the way they think about the quality professionals' role, test technologies and processes. I will also share some real- life experience from my own work environment, as it is undergoing massive changes in its development methodologies and test technologies to prepare for the inevitable.

The constant challenge of quality

It is obvious that we are all challenged with the need for quality in the work we do – sometimes more, sometimes less. But the notion of quality is one where it is actually hard to find consensus. What does it really mean? Depending on who you ask, you will get an answer like "I know it when I see it (but I don't see it often)" from an end user or "I can write high-quality code (and I wish I could assure that mine works with the rest of the build all of the time)" from a developer. I like to see quality as the convergence of at least the following: complete requirements, correct code and minimized defects that align to meet business goals. Even if we can't all agree on a definition like this - one thing should be clear nevertheless: Quality is not a discrete or isolated function, it is an ongoing and cumulative effort that not only involves testers but many parties – even though often they may not be aware of it. So when it comes to making sure that quality is up to the standards we like to see in software development, software testing is the activity used to help identify correctness, completeness, security and quality of the computer software developed. Of course, testing as such does not equal quality nor is it possible to test quality into a product - but testing it is a crucial activity of the life cycle quality process!

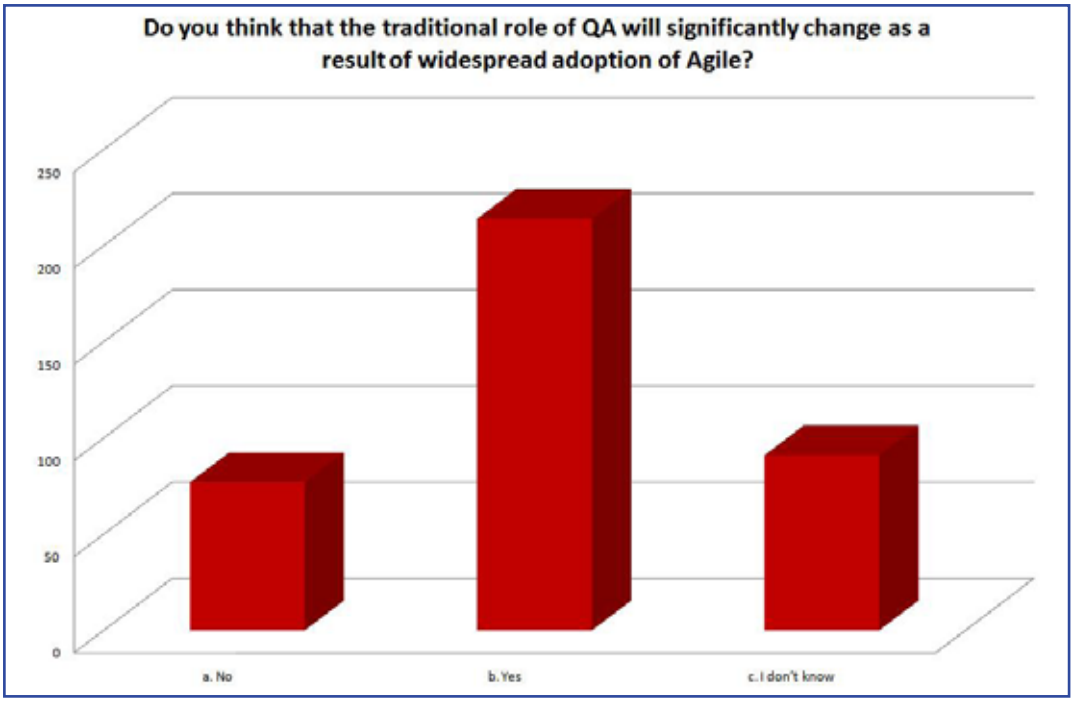
Familiar problems

Today we are very familiar with the problems in software testing because we feel the pain they cause in many ways. Typical problems include late lifecycle testing which usually means that the majority of bugs are found late in the development cycle when they are extremely difficult and expensive to fix. Limited testing - often in combination with a very high percentage of manual tests or a heavy bias on unit testing - results in minimal functional testing and almost no performance testing and other non-functional testing activities like security or vulnerability testing. One of the most well known yet most common issues is related to poor quality requirements which drive complete projects and with it the testing. Changes which are not reflected in testing will inevitably lead to misalignment with business needs.

These are just some of the problems we still see today, but the pressure to overcome them increases every day – especially in a challenging economic climate. One obvious way to counter these pressing problems is to invest in education and elevate the status of the testing discipline and its main actors from what many consider to be a necessary evil to being the bright light of software development!

Evolution of the QA professional

Let's start having a look at the trends of the future by first examining the effects these problems have had on the most important element involved in the process – the human. Looking at the traditional roles and responsibilities in testing, there have always been a number of clearly distinguished personas that play an important part in the testing game. Examples include the Test Manager who is the quality and test advocate responsible for resource planning and management as well as the resolution of issues. The Test Analyst identifies and defines the required tests, monitors testing progress and evaluates the overall quality experienced as a result of testing activities. This role typically carries the responsibility for appropriately representing the needs of stakeholders that do not have direct or regular representation on the project. The Test Designer defines the



test approach and ensures its successful implementation including identifying the appropriate techniques, tools and guidelines. And the Tester usually implements, sets up and executes tests, logs outcomes and verifies test execution, analyzes and recovers from execution errors. A tester should have knowledge of testing approaches and techniques, diagnostic and problem-solving skills, knowledge of the system or application being tested, and knowledge of networking and system architecture. Of course, there are additional roles and more often than not there is an overlap between these.

New testing paradigms

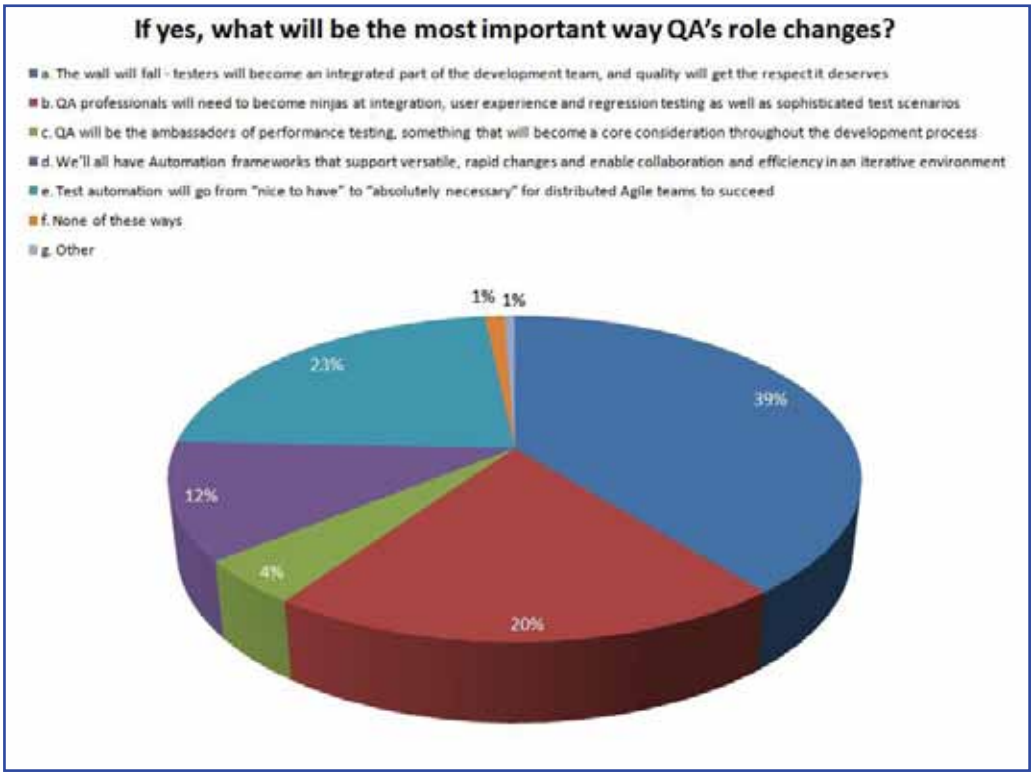
However, over the years there have been quite a few quality issues that made it into the news – with some of the most prominent ones happening in recent history. So today there is a much higher visibility of quality - or the lack thereof - in the public and a higher consciousness for it. With it came some rethinking of the established roles in testing. Today there is a much higher need for alignment of business needs and engineering needs from the start. There is a strong demand to build in value from the very start of each product development. It is no longer enough to just focus on developing code, but there is the requirement to look at the full lifecycle of a product. Obviously, this starts to have an impact on the testing approach. Today and in the past we have seen what I like to call engineering-heavy approaches where not much outside of the immediate code development activities was considered from a quality perspective – if that! Still an incredibly high number of software projects involve no formal testing at all!

These days, however, we see an emerging approach of a risk-based/quality- conscious view, which takes into account additional parameters from outside the engineering world. Business needs begin to drive quality requirements more directly through a stronger connectivity and traceability between requirements, developed features and required tests. With this comes a stronger demand to deliver increasing value in highly specialist skill areas, such as Test Automation, Performance Testing and Security Testing. In the same way as quality becomes a much more important aspect of the software delivery lifecycle, so does testing become a crucial activity in the lifecycle quality process. It drives the need for more skilled QA professionals that know how to collaborate and build more advanced and larger test sets.

This is especially true in agile environments. In the last few years we have seen a dramatic increase in agile approaches, and these are one of the most important agents of change. Interestingly enough the agile literature does not say much about testing, but anyone who has been involved in agile projects will quickly realize that traditional testing approaches will not work particularly well there. One of the first thing agile teams will recognize is the fact that test automation will be indispensable in such environments where short sprint cycles – more and more teams are adopting two-week sprints - are becoming a key element of the development strategy. And test automation does not end with unit testing! This means that there will not only be a much stronger demand for test tools that can support a high degree of automation, but also a need for highly skilled, much more technically competent testers. The days of purely manual testing or simple click testing tools that – supposedly – require little or no technical knowledge from a tester are definitely over. Roles will shift as part of an agile transformation, and the agile wave is something of a wake-up call for testers. Testers, get your skills ramped up!

How will QA's role change?

In a recent survey carried out by Borland 56% considered that the traditional role of QA will change, and significantly so. These respondents were then asked further, what will be the most important way in which the role of QA will change.



By far the greatest number (39%) believed that the concept of an isolated QA team will disappear and that instead, QA will become an integrated part of development. It was also interesting to note that the second largest response was that there would be a change in perspective on test automation, from 'nice to have' to 'absolutely necessary' (23%). So despite the earlier reservations from respondents about current tool offerings, the need for tool automation is still seen as a priority in order to ensure quality software.

Expanding the tester skill set

While it is obvious that roles in QA will shift, it doesn't mean that core responsibilities will go away. For example, while there are many things that are different with Agile, the actual testing activities – and the challenges with those – are not different. It will still be necessary to identify the most appropriate implementation approach for testing. Also, implementing, setting up and executing individual tests and logging the outcomes and verifying test execution results or analyzing and recovering from execution errors will all remain necessary activities. Both Agile and traditional teams need to effectively test their software for functionality, performance and scalability. The difference lies in the application of these activities – the timing, the workflow, the assigning of activities – all these things are different in Agile.

Even accomplished testers must seek to expand their skill set to include virtues like strong collaboration capabilities with others, e.g. developers as cultural aspects as a whole# become much more important. It will become especially important to adapt to agile development practices and become a member of "the team" as the Dev/Test barrier gets removed. But it doesn't end here. There is also a strong need to embrace new technologies like modern testing frameworks (e.g. Fitnesse, RobotFramework, etc.) as well as popular open-source and powerful commercial tools that help to master the additional challenges of testing in the new world. Most importantly – and many a tester will not like this perspective – it will become important to develop programming skills and get involved in test conception from the beginning. In agile projects one thing, that many of us have suspected for a long time but never dared to speak about in public, becomes obvious – testing is as much a development discipline as is the coding of the application that needs to be tested! As a matter of fact, both go hand in hand and developers and testers can only benefit from close collaboration and similar (coding) skills. Ultimately this is only a manifestation of the fact that in any software development project everyone is responsible and accountable for quality.

Test technology trends

Having considered some of the changes which are beginning to shape the lives of QA professionals in the future, let's focus on another important aspect. Software testing has a long history of technology which promised to make the job of the tester easier. Unfortunately, this promise has only been partially fulfilled in the past, giving tool support for testing a bit of a bad name in the industry.

Test automation

It has been around for quite some time, mostly in the areas of performance, scalability, reliability, and stress testing as well as functional and regression testing. It cannot always be applied, but even where it is applicable, it still offers great potential today. There are several reasons why over the years test automation has really only had limited success and not lived up to its promise. Early versions of test tools were not particularly mature and required a substantial amount of effort to build robust test automation sets with them. At the time, skilled test automation experts were rare, and inadequate usage of tools didn't help to improve things either. Additionally, technologies evolved quickly and tool vendors sometimes had a hard time keeping up with market and technology trends.

However, tool vendors have learned their lessons and tool sets are maturing. Of course, tools need to be used in the right context to achieve maximum effect, and there have been great improvements with regard to usability and efficiency. Again, trends like lean or agile development have put more pressure on vendors to improve the robustness, speed and ease of use of their products. These new trends will offer further potential in the future, as does outsourcing or global distributed development which all put high demands on testing tools. Not only is performance testing impossible without automation, today and in the future other types of testing like functional testing, regression testing or acceptance testing will all require a high degree of automation if carried out as part of a rapid development scenario where testing is highly integrated.

Open source

For a while open-source tools have been regarded as the remedy to the many problems in testing that commercial tools apparently were unable to solve. Most often the argument that open-source software is free has been used against commercial software. However, "free" can really be expensive, as many have seen in their projects when costs of maintaining and extending very limited sets of functionality have grown exponentially. Actually, open-source software very often doesn't live up to its promise either – it is brittle, unstable, very limited in its use and usually difficult to install and administer. Also, lots of open-source projects are abandoned and there are countless tools out there in the web. Which is the one to choose?

On the other hand, commercial tools have always had their challenges, too. Vendors found it hard to keep pace with changing technology and provided limited support in certain areas – even though usually far more comprehensive than open- source alternatives. Commercial offerings curiously don't always do a good job of facilitating collaboration, which is becoming increasingly important in modern integrated development scenarios. Most importantly, a relatively high cost factor is often seen as prohibitive for a large scale use in an organization. However, that really depends on how much it is really used.

I believe that it is really not one or the other – commercial or open-source software. As a matter of fact, commercial and open-source tools will integrate a lot more in the coming years and open source will even become absorbed in many commercial tools. And in projects it really makes sense to use the tool that fits best!

Automation tools and technologies they support

Actually, there is a far more interesting distinction than that between open-source and commercial tools. On the one hand there are what I like to call specialized tools for specific technologies, and on the hand there are tools covering multiple technologies. Let's have a look at the first group. This group includes tools that specialize in testing more or less just one area of technology and with it applications built using that technology. Typical examples include tools for testing Web apps, tools for testing Java apps (AWT, SWT, Swing, etc.) or tools testing .Net (WPF, Winforms, etc.) apps. There are examples for Win32 or custom technologies. While these tools usually do a pretty good job of testing applications built in that particular technology, they are pretty much useless for anything else. However, applications often have mixed technology. For example, Web apps might have Flex or Silverlight technology embedded, Java might have IE controls embedded or a .Net app might have custom controls embedded.

What does this mean for testing? In such cases testing requires either multiple (potentially incompatible) tools or a mixed approach of manual and automated testing, both of which is far from desirable. Also, the replacement of technology will leave test sets useless, which is a real problem. Nevertheless such tools will continue to be around and they will be useful in certain cases. However, none of them is likely to become main stream for the reasons listed above, even though sometimes there will be initiatives to extend them to support additional technologies.

For mixed technology applications such tools provide a seamless testing experience and allow for much more holistic (and more realistic) as well as more robust testing. There is a much higher chance that less or no manual testing is required and, more importantly, technology change usually leaves test sets unusable in the future. Of course, for single technology applications they provide all of the above and they can usually be reapplied for other technologies quickly. As a consequence these tools will become more sophisticated and will become much better integrated with other tools to support collaboration.

Test development and integration

If we look at the overall integration of testing tools and their support for test development, we will see a couple of trends in the future. There will be a concentration on core competencies by vendors in the areas of functional testing, load testing and test case generation and management. It will become very important to leverage and incorporate existing solutions like test sets, testing frameworks, test tools and additional technologies. There will also be a better integration with test management solutions as well as business management solutions. It will become important to support collaboration eliminating the drawbacks of working in silos. Keyword-driven approaches will become much more dominant to better support the building of large-scale testing frameworks.

Test management

Test management tools will begin to play a much more important role as organizations mature beyond the stage of manual testing and simple test automation. They will require a consistent way to manage and report on their test assets and overall progress. In the future we will see support for different processes like waterfall, iterative and agile. There will also be a much tighter integration with requirements management systems, testing tools (open-source and commercial), source control systems and testing frameworks and agile team tools. Another important aspect will be the convergence with Business management systems, in order to provide valuable metrics and increased visibility in the management dashboard. All of this brings together different aspects of testing like requirements management, test planning, test execution, code coverage, defect tracking, metrics, and source control. There will be solutions that provide visibility on quality metrics to various stakeholders from management, QA and developers.

Testing in the agile world

One of the most important drivers in the change of software testing is certainly the continued rise of the agile wave. I have already given some examples on how this will have an impact on the future of software testing. I would like to expand on some aspects of this a bit more, as I believe this to be one of the biggest challenges – and opportunities – for QA professionals as well as test tool vendors and service providers alike.

Agile development and testing

There are a number of best practices that are commonly associated with agile development. The idea of a generalist approach which promotes generic vs. specific skill sets that are scarce, daily kick-offs and reviews of goals, short release cycles and responsive development are just some examples. While none of these are really new concepts, it is really the framework of the agile manifesto and much of the related information that was produced around it that turned out to be a catalyst for many of the changes we see in development today. However, it should be noted that at the same time this has lead to some illusions about the state of development in many organizations. Just because a team follows some of these principles, it doesn't necessarily mean that they are agile!

More importantly, however, agile isn't so specific when it comes to testing. Again, there are some principles that provide guidelines here. For example, the idea to concentrate on developing and testing high-value features first or the notion of test or behavior-driven development are common with agile development strategies. As explained earlier, such agile strategies will require a higher percentage of test automation for unit, functional, acceptance and performance tests. And while there are little specific guidelines in agile literature regarding these activities, very often the sheer necessity to improve things here will drive action. Other examples include strategies for continuous build and integration which start to extend into testing beyond the unit testing stage. Finally, it should be noted that the concept of pair programming of developers and testers will become even more important in the future.

Test automation - speed and repeatability

In agile scenarios where compacted sprint cycles are a manifestation of a high- velocity approach that includes coding, documentation and testing, there is a strongly increased need to accelerate the code-and-test process by supporting fast, reliable and low-maintenance automated test scripts. The faster test scripts can be executed, the more tests can be run resulting in better test coverage. And the faster tests can be run, the more often build verification test sets can be run as part of the build cycle. Teams need to guarantee the uninterrupted repeatability of tests to ensure regression testing from sprint-to-sprint or iteration-to-iteration. Also, there is much more need to enhance test efficiency further via robust, yet flexible test management processes and avoid the inherent inaccuracies that manual processes inject into a process – particularly when time is tight. The goal is to lighten the workload of testers and eliminate the need for late night and weekend testing marathons that can burn teams out. Ideally, large test sets can be run with each build allowing immediate feedback to developers about issues that have been introduced in the code.

Test often and early

The "test often and early" principle also gained a lot of popularity with the advent of the agile wave. It provides a means for early feedback to developers about the quality of their code. It is important to note that in the future it will be required to extend the information which is fed back to the engineering team beyond what is typically offered as part of unit testing activities. Performance trend information across builds that includes transaction response times, page times, and other custom measures is just one example of how a more comprehensive approach to quality metrics will shape future development and testing efforts. One of the important characteristics of agile development is that new features of an application must not only be coded but also tested and documented. This is one of the major drivers of the "test often and early" principle.

Improving the testability of applications

The often limited success of test automation in the past has not only put pressure on test tools vendors. More and more organizations seek to develop applications with testability in mind, as can be seen in the rapid adoption of test-driven development approaches. Not only does this mean that the questions of what and how to test an application are being considered early on. It also means that a lot more thought is given on how an application can be enhanced to make it more testable. If good testability of an application is complemented by a test tool set that can leverage such enhancements, testing becomes dramatically easier. There are a number of ways how testability can be improved, but the options really depend on whether an application is designed with testability in mind from the very beginning or whether it will be enhanced at a later stage in the lifecycle. The latter is usually the case with older applications that also will require larger regression test sets. Instrumenting existing interfaces with testability hooks and adding attributes that can be used for testing are typical approaches that can be used here. Testability hooks make it easier for testing tools to understand the interface from both tool's and tester's perspectives, consistently recognize and call actions and verify actions and responses. However, it is important to understand that this will require a collaborative effort between testers and developers and usually the involvement of an architect. So once again, collaboration amongst various stakeholders will become important here to reap the benefits of such an approach.

A lab's agile transformation

To illustrate the points I made above, I would like to share some of the experiences of the engineering teams based in Linz, Austria that are working on Micro Focus's Silk product line of testing products. They have seen massive changes in their development methodologies and test approaches as part of the agile transformation that was started more than two years ago.

For each of the products developed in Linz, there was originally one dedicated team of developers. They were responsible for implementing the features, which were described in a Product Requirements document which was owned by the Product Manager. The requirements were usually described in a few lines and often very high-level. Generally, there were no use cases or information about relevant configurations. Moreover, no attention was given to the testability of a requirement. Consequently, versioning and tracking of decisions or changes was hard, and changes were often not communicated. The developers had

time to implement the features until the "Feature Freeze" milestone. At that time the features were handed over to the QA team for testing.

QA was originally a dedicated group shared between the three products. Once the developers were finished with development, the features were in a state called 'QA Ready'. This was when the QA team started to execute the test cases, which they had described in a big test plan. The developers' thinking from that point on was that they were "finished" with the feature. 'QA Ready' meant that the feature was unit-tested and there was a development paper with a feature description and hints for the QA. The test plan was generated based on the information provided by the Product Requirements document and additional development papers. Obviously, there was a problem. The test plans usually weren't in-sync anymore with the implemented functionality due to changes made during the course of development and a lack of communication.

Often development was behind schedule and they had really only had two options: Give a piece of code that had been implemented to QA, even if the feature was not really 'QA-Ready', or take more time to bring it to the required level of quality. In both cases it was bad for QA. Either they had less time to execute the tests, or they found a lot of bugs due to lesser quality of the code. We also had a dedicated documentation team. Again these were shared resources which were responsible for documenting the features. The doc team was in a similar situation like the QA team. Changes didn't really come through to them and features to get documented often arrived late.

The first area of improvement was identified in how product management should define requirements in the product requirements document. Often the Product Manager was not available for answering questions or improving the quality of the requirements, and many times changes were not reflected in the product requirements document. We overcame this by introducing a new role, the Product Owner, and using a dedicated Requirements Management System.

We then had multiple Scrum teams per product, and we would usually also need additional product owners, because each team should have their own product owner. But due to limited resources we were only able to assign one product owner per product. This, however, also has some advantages. Having more than one product owner per product introduces other problems. For now we go that way and we will check after a while if this is an impediment for the teams. We have regular retrospective meetings in place, and this will help to identify this immediately.

Summary

Let me try and summarize some of the key points introduced in this article. One of the most interesting trends we are starting to see is that testing is finally becoming more aligned with business needs. Strategies like test-driven development are a manifestation of this, but there is still plenty of room for improvement. There is a growing understanding that quality will become everybody's responsibility in the future, and more and more organizations start to look at quality more holistically. However, again we are just seeing the very beginning of this right now. With agile development strategies and the faster development cycles that come with it, test automation will become much more important. Without good, robust test automation it will be impossible to keep quality up, let alone improve it in such environments.

We will also see a concentration on core competencies by test tool vendors, which will come hand in hand with tighter integration of toolsets and leveraging of third-party tools – both open-source and commercial. Adoption of the "test early and often" principle will become mainstream, and there will be a stronger need to improve the testability of applications to support this. Finally, developers and testers will need to collaborate much more in the future, which will require both groups to ramp up their skill sets in other areas. Any process of transformation will need to be monitored to make sure progress is being made and adaptions are applied when necessary. The future certainly will be interesting for testers and everyone involved in the overall quality process.

Biography

Joachim Herschmann is the director of product management responsible for Micro Focus' Silk line of testing products recently acquired from Borland. At Borland Joachim was in charge of Borland's Lifecycle Quality Management (LQM) suite of products. Previous roles at Borland included solutions marketing manager LQM for the EMEA region. Joachim came to Borland in April 2006 through the acquisition of Segue, a leading testing solution provider. While at Segue, Joachim spent several years as technical account manager and technical consultant in the area of software testing and quality assurance. Joachim has more than fifteen years of IT-business experience — more than half in the consulting and testing business.

© Katrin Schilling

3 day testing course by Alon Linetzki

November 04-06, 2009 in Berlin,Germany

Limited places

Adding Business Value & Increasing ROI in Testing

Introduction
We are facing today an increase demand from test managers to improve productivity and product quality, to reduce costs of testing, to reduce resources, and to make-things-right-the-first-time.
This challenge is forcing test managers to improve in 3 major topics:

- Improve testing processes
- Manage testing project risks better
- Manage professional testers, and the supporting environment better by improving communication, diplomacy and negotiation skills

The course is enforcing those and aiming to educate test managers in how to be better in these areas of concern. The course includes practical exercises and simulations in the relevant topic areas.

Description
The learning objectives of Adding Business Value & Increasing ROI in Testing course focus on the below main areas of practical doing:

- To be able to know what test process improvement and TPI® model are
- Discuss a method for evaluating process maturity from different aspects
- Discuss the different dependencies between the testing processes
- Know how to quick-start TPI® effort in a testing organization
- Discuss which projects to pick for the TPI® pilot (with best chances to succeed)
- Understand the implications of managing risks in testing projects
- Understand the concepts of Risk Management
- Describe Risk Based Testing principals
- Understand what is the Risk language
- Define where RBT can assist during the testing life cycle
- To learn how to exercise an **excel tool** for risk strategy (planning phase)
- Discuss test execution strategy issues and RBT
- To learn how to exercise an **excel tool** for risk analysis (scheduling and execution phases)
- To learn about good communication concept and methods
- To know different paradigms of personalities, and reflect those on us
- Understand how to convince others without getting resistance
- Discuss how to present and pass a message to others
- Discuss how to give and obtain feedback, and get a positive impact

For more information and to register go to:
www.testingexperience.com/knowledge_transfer.html

1450,- EUR
(plus VAT)

te testing experience
Knowledge Transfer

Díaz Hilterscheid

TPI® is a registered trademark of Sogeti Nederland B.V.

Testability: Investment, not Overhead

by David Evans

Every application can be tested, but not with equal ease. Every application should be tested, but not with equal intensity. As any risk-based tester knows, given a limited amount of resources for testing, you should concentrate those resources on testing areas of greatest business risk. Unfortunately it is often the case that the intensity of testing effort across an application mirrors the ease of testability of the application. Put simply, stuff that is easy to test gets tested more than stuff that is hard to test.

The trouble is, more often than not, the areas that are hardest to test are exactly the dark and dangerous alleys where defects and business risks lurk. As Kent Beck [1] said, "Code that isn't tested doesn't work – this seems to be the safe assumption."

The easier we make an application to test, the more it will be tested, leading to a better quality application. *Testability* is the measure of how well and how efficiently an application can be tested. Designing an application for testability is not just a lofty architectural goal; it saves time, money and risk. Moreover, from a software architecture viewpoint, a testable application is inherently a better application. The prerequisites for testability are congruent with the characteristics considered to be good architectural practice.

A testable architecture implies at least these characteristics:

- Clear separation of concerns (multi-tier)
- Loose coupling
- High cohesion
- Maintainability
- Installability
- Upgradability
- Flexibility
- Extensibility

Note that *testability* has a very close relationship to *maintainability*. Although both qualities have slightly different 'interested stakeholders', they are both improved by very similar architectural characteristics. Maintainability lowers the total cost of software ownership. Testability lowers the total cost of software quality.

Total Cost of Quality

Application Testability is an architectural or design **cost**, incurred for a testing **benefit**. To get the most benefit later, we need to make the right decisions earlier. Understanding the business case is the primary prerequisite for making an effective investment in Testability.

Testability improves (at least) these aspects of testing:

- Repeatability – tests can control their preconditions and produce the same outputs given the same inputs on repeated execution
- Isolation – each test can specifically and reliably assert the behaviour of one area of the system, without being affected by the behaviour of other areas
- Speed – tests can be automated to provide very rapid feedback

All these aspects lead to faster fault identification, isolation and diagnosis. A large suite of fast automated tests, executed often, comprising tests with clear and specific assertions, becomes a powerful weapon for detecting and removing errors as near to the point of injection as possible. This greatly improves overall test efficiency, lowering the cost of quality.

Considering how to improve testability in application architecture and design generally leads to the need for these features:

- Programmable interfaces
- Dependency Injection
- Detailed trace logging
- Clear error messages & exceptions
- Unique identifiers for controls, exceptions etc.

These undoubtedly carry a certain design cost, but are generally accepted to be long-term benefits over and above the improvements to testability. For example, it has been reported [2] that the need to create an API (application programming interface) for testing purposes on early releases of Microsoft Excel ultimately led to exposure of the public VBA macro interface that is now considered by many to be one of the best features of that application.

Time Travel

Every application has its particular testability challenges, but there is one thing common to most business applications that is always hard to test: time. Most systems have some form of time-dependent processing or triggers, whether it is calculating overdue payment status, end-of-month processing, task reminders, automatic record archiving, bank-holiday exceptions... the list goes on.

A good tester does not wait to the end of the month to test month-end processing. They grab the bull by the horns and the clock by the hands and they bend time to their will, like some mischievous minor deity. They write automated tests that go along the lines of "Order an item on Jan 1, check payment due Feb 1, Pay invoice on Feb 10, check 10-days overdue interest charged" and they get the answer back in seconds, not weeks.

Such testing is easy if the application is built with a 'mockable clock' [3]. This means that for testing purposes, the tester can simulate or 'mock' the time and date that the application logic uses. There is no need for any exceptional coding within the application logic, it just needs to have a central internal source for getting the time (instead of looking at the operating system time), and a separate mechanism for allowing a test to override the 'real' clock with a mock clock. This typically means implementing methods such as:

```
•   GetTime () // Central function used by the system
•   SetTime (mockTime) // Set fake time for tests
•   ResetTime ()       // Revert to real time
```

Faking the clock for testing purposes is just one example of the use of mocking to control dependencies and non-deterministic test conditions. There are many sources out there for details of mock object frameworks for unit testing, where test isolation is a fundamental principle and mocking out dependencies is a good technique. But such frameworks are not needed to achieve the basic testing need of controlling the application's sense of time.

Testability and Software Development Methodology

A mature Agile software development organisation is likely to have mixed-skill teams of developers and testers, sharing responsibility for test automation, utilising Test-Driven Development and regularly running large automated test suites using a Continuous Integration solution. For such teams, built-in testability is not so much a matter of choice but an obvious necessity. Creating automated tests is considered part of the cost of iteratively developing each feature, so the cost of testing looms large throughout the project, and the whole team is motivated to reduce it.

Contrast this with the situation of a large waterfall project. The testability of an application affects those conducting testing at the end of the project life-cycle, long after the opportunity to improve or design-in testability has passed. Even if the issue of testability is considered during design, there is little motivation for architects to seriously embrace this if testing is still a distant activity on the project timeline, and the testing strategy has yet to be clearly defined.

Most projects exist in an uncertain, pragmatic middle ground. Not every agile project adopts a pure test-driven approach. Not every waterfall project divides testers from designers so distinctly. Regardless of your situation, it must be recognised that the greater the organisational or communication divide between testing and development, the harder it will be to make the case for testability. Outsourcing development and testing to different organisations is the extreme case of such a divide, and is likely to make testability considerations very difficult to assure.

Taking full advantage of testability often requires 'opening' application design. Testers get visibility into application design, and developers get visibility into test case design. Thus, concerns are shared and the whole team gains an opportunity to learn and benefit from the experience of different parties. In an environment where there is a healthy mutual respect and attitude of co-operation between architects, developers and testers, testability will be seen to be a common goal.

In James Bach's heuristics [4] for testability, three interesting categories emerge:

- Technical heuristics:

  Control (allowing repeatable or automated processes to control the system)

  Observability (allowing system state to be queried by tests)

- Process heuristics:

  Simplicity (doing only what is required, no more)

  Stability (ensuring reliable and consistent behaviour, without regression)

- Communication heuristics:

  Availability (ensuring testers have good access to the system during development)

  Information (ensuring testers have full knowledge of required or expected behaviour)

This highlights that testability is not only about the structure of the application under test, but about the structure of the team responsible for developing it.

Summary

Testability is the extent to which a system or component can be tested efficiently.

Improving testability is an up-front investment with many long-term benefits, including:

- Lower Cost of Quality
- Better software architecture
- Improved maintainability

Strategies for improving testability include:

- Architectural: 'opening the black box'
- Technical: utilising test support tools like mocking frameworks
- Procedural: ensuring the development process supports testing
- Human: understanding the needs, skills and motivation of testers, developers and other stakeholders

References

1. "Test Driven Development" by Kent Beck. Addison Wesley, 2002.
2. "Design for Testability" by Bret Pettichord, http://www.io.com/~wazmo/papers/design_for_testability_PNSQC.pdf
3. "The Virtual Clock Test Pattern" by Paolo Perotta http://www.nusco.org/docs/virtual_clock.pdf
4. "Heuristics of Software Testability" by James Bach, http://www.satisfice.com/tools/testable.pdf

Biography

David Evans is the Agile Services Director at SQS, Europe's largest pure-play quality consultancy. With over 20 years' experience in software development and testing, he has had several papers on software testing published in international business journals and is a regular presenter at software & quality conferences. David is a thought leader and evangelist on Agile testing, and has consulted on this topic for organisations in the UK, USA, Europe, India and South Africa. For two years he led an agile team developing and testing the award-winning application "TestStrategist" for SQS, which became the subject of a case study in John Watkin's forthcoming book "Agile Testing".

© iStockphoto.com/Elm

testing experience

Knowledge Transfer

— TMMi® Foundation —

One Day Tutorial with **Erik van Veenendaal**

limited places

NEW DATE:
November, 5th 2009 in Düsseldorf, Germany

This workshop brings the TMMi® model to Europe and is your chance to learn about the latest initiative in test process improvement.

The Test Maturity Model Integration is rapidly growing in use across Europe and the USA. It has been developed to compliment the existing CMMI framework. Its growing popularity is based upon it being the only independent test process measurement method, and the simple presentation of maturity levels that it provides.

In Europe the independent TMMi® Foundation initiative has been established with the sole intent of elaborating the TMMi® standard and developing a standard TMMi® assessment and certification method, with the aim of enabling the standards consistent deployment and the collection of industry metrics.
Erik van Veenendaal has much practical experience in implementing the model and helping organisations improve the way they test, and the benefits this can generate. He is the editor of the TMMi® Framework model and vice-chair of the TMMi® Foundation. The workshop will present these experiences and benefits with the aim of providing the attendees with the information required to justify a test process improvement project.

As well as learning more about the TMMi® during the workshop each attendee will be provided with the information and practical materials needed to do an evaluation of their own companies test maturity level.

Key learning points

- Understanding the objectives and (intermediate) results achieved in the TMMi® Foundation
- Understanding of the TMMi® model and its practical implementation
- Practical application of the TMMi® assessment techniques

For more information and to register go to:
www.testingexperience.com/knowledge_transfer.html

750,- €
(plus VAT)

TMMi® is the registered trademark of the TMMi Foundation

© iStockphoto.com/Andyd

Driving an Agile Peg in a CMMI Hole

Testing in a Quasi-Agile Environment

by Timothy Korson

While we were sitting around the breakfast table this morning, my wife asked me what I had to do today. I replied that I needed to write an article on "Agile Testing". Overhearing this conversation, one of my daughters quipped:

"Agile testing??? You mean as opposed to clumsy testing?"

As I think about, I realize that her comical comment is, in fact, accurate. The way I have to test on a traditional project is rather clumsy and inefficient compared to the way I test on an agile project. I admit it. I am an agile proponent. I favor an environment where the management team, the development team and all of the stakeholders are on board with agile development. I have found that the team can produce systems not only faster, but of higher quality, when the team I am working with uses mature agile methods. But I am also a realist. Often I am faced with an environment where corporate policies and procedures prohibit the team from using agile methods "by the book." I could choose to not work on those projects until the organization is fully agile compliant, but I find that there is value in helping software development teams "drive an agile peg in a CMMI hole."

The reality is that most corporations are still fairly traditionally structured even though many software development teams are heading full steam into modern, highly iterative, agile software development techniques. This leaves management stuck coping with an organizational and technical paradigm shift that traditional project management practices are inadequate to handle. In the highly iterative, fast-paced environment characteristic of these modern software development projects, traditional approaches to testing, quality assurance, requirements gathering, scheduling and estimating break down. Managers trying to encourage best practices as recommended by CMMI and SPICE find themselves at odds with developers trying to adopt best practices as recommended by the agile manifesto. This article discusses practical ways for testers faced with the formal, heavy weight, process control inherent in CMMI recommendations to still achieve many of the lighter weight, more flexible practices of agile development. The goal is to produce a pragmatic, yet productive quasi-agile development environment.

In this article, I am not addressing the many additional issues that arise when an organization tries to be both agile and officially CMMI certified. Instead I am addressing the simpler question of how an agile tester can survive in a traditional organization where many policies and procedures are derived from the waterfall process and the CMMI philosophy.

I find the following three simple principles helpful in succeeding in a quasi-agile environment.

1. An attitude of multi-cultural tolerance which allows for the peaceful co-existence of processes derived from very different philosophies.
2. Barely sufficient compliance.
3. Focus on the goal of traditional control structures and show how the goal can be met in innovative agile ways

The following 2 policies are typical for a traditional organization, but not suitable for an agile development team. I will use them to illustrate how the above principles can be applied.

- All issues must be documented in an issue tracking system.
- There must be a detailed comprehensive requirements document.

All issues must be documented in an issue tracking system

In the ideal agile environment, when a tester discovers an issue, the issue is discussed with the developer face to face and resolved in as short a time frame as possible. The issue is only logged and tracked (perhaps with a yellow sticky note) if it cannot be resolved relatively immediately. Compared to the inherent problems of non-reproducibility, inefficient use of time documenting, and endlessly cycling back and forth between: opened, not reproduced, re-opened, fixed, reopened, fixed, disputed, re-opened… the agile approach does indeed make the traditional approach seem clumsy. What do you do, however, if your organization insists that all issues be logged in the issue tracking system?

1. Approach the problem with an attitude of trying to understand the manager's point of view. Perhaps the manager uses that data to justify budget requests. Use the other two principles to work out another way to justify the budget, but in the meantime, realize that the manager has a job to do too. If agile testing makes that job apparently harder, then you owe a bit of tolerance to your manager.
2. Use the issue tracking system in a barely sufficient manner. Use the agile, face to face, approach to resolve the issue; then after the issue is already resolved and re-tested, take a minute or two to create an issue in the tool, add a minimum of comments, then close the issue. Issues that are more complex, or represent higher risk, can be logged with more detail. For very minor issues, like correcting a misspelling on a screen, just stop logging them. Don't ask. Don't tell. Just stop.

   Sometimes, instead of barely sufficient, look for more efficient ways to satisfy the requirement. Personally, I find that the most time consuming, inefficient part of using an issue tracking tool is logging the steps required to reproduce the issue. I have recently started using a screen capture tool, like Jing[1], to capture and voice-annotate the steps needed to reproduce an issue. In an agile environment, I can easily capture the session while I am showing the developer an issue. This recording can easily be logged with the

issue. Thus the organization has a detailed record in the tool, but by thinking of more efficient ways to satisfy a process requirement, I am incurring barely more overhead than the simple agile process.

3. Find out all the reasons you are required to use the issue tracking tool. For each reason try to find an agile alternative.

Suppose as suggested above, that your manager really does use reports from the issue tracking system to justify budget requests. Likely, the manager uses only summary data. You may be able to use a simple spreadsheet to give managers the data they desire. Or, as suggested above, a minimally sufficient use of the issue tracking system may suffice to generate summary reports. As the quasi-agile approach is seen to work, hopefully your manager will be willing to explore an agile approach to management and won't be using issue tracking data to support budget requests.

There must be a detailed comprehensive requirements document.

Agile teams tend to be small. Often there is no business analyst on the team. The client tells stories. The team elaborates the stories into an executable system and the only documented detailed requirements are the set of executable system test cases. But what if management insists in an up-front detailed requirements document?

1. Again, realize you're not the only one trying to get their job done. It is very possible that the manager can't, or doesn't know how to, get funding without a requirements document. Not everyone has to adopt agile techniques simultaneously. It is quite possible to have an agile development process co-exist peacefully alongside of a management process that retains a number of waterfall elements. Clearly this is sub-optimal, but it can work, and is often the only way to get agile techniques integrated into the larger organization. This is an important point, so I want to repeat it: *Not everyone has to adopt agile techniques simultaneously. It is quite possible to have an agile development process co-exist peacefully alongside of a management process that retains a number of waterfall elements.* For this to happen all parties must be comfortable with the arrangement, only then can the organization successfully work through the issues of how to make it happen.

2. Create a barely sufficient requirements document. Instead of putting as much detail in the document as possible, put in as little detail as reasonable. Focus on scope, characterizing complexity, and capturing those elements of the requirements that the stakeholders are the most sure will not change. Put in just enough detail so that the manager can make a first guess at cost and schedule (number of iterations required).

3. Determine why a requirements document is needed and then demonstrate the agile approach to meeting those needs. This can be a difficult task that requires a high level of skill in conflict resolution. I recommend that anyone involved in process change read the literature and take classes on topics such as conflict resolution and organization behavior. I personally like Goldratt's "evaporating cloud" approach [2] to these issues.

Most managers want a requirements document to be the basis of a contract in order to control scope. In the agile world scope is the variable we don't want to control, so we have an obvious conflict here. Therefore we have to back up a level and ask managers why they want to control scope. For some managers this is a very unsettling question. In their paradigm, scope is an axiom, a matter of first principles. There can be no why about scope, because without a requirements document there can be no project. Fortunately the agile community has already thought this through. We can help managers realize that what they really want to control is the cost and schedule that will be needed to meet the stakeholders' business objectives. We then have to explain the agile approach to controlling cost and schedule without freezing requirements [3]. In a quasi-agile world this could entail extra work, as the manager might have to present somewhat of a waterfall front to part of the organization and an agile front to the development team.

Another reason managers want to have a requirements document is as a basis for creating a comprehensive set of test cases. In mature agile approaches, the agile tester takes on the additional role of business analyst and captures the requirements directly from the stakeholders. In the most extreme agile approaches, the only complete, detailed documentation of the requirements is a set of scripts that automate the system tests. In a quasi-agile environment the test scenarios, test scripts, and test cases take the place of a traditional requirements document and will have to be documented in a form that stakeholders can read and understand. The test design document may even be the mechanism for client sign-off on requirements. This sign off would occur on an increment by increment basis.

To summarize: In this case the "CMMI hole" is a complete up-front requirements document. The "agile peg" is an incrementally developed set of test scenarios, test cases, and test scripts along with a set of user stories and a minimally sufficient scope level requirements document. Overriding all of this is an attitude of understanding and appreciation. The managers appreciates that the agile tester is dedicated to helping the team deliver high quality code as fast as possible with as few resources as possible. The agile tester appreciates that the manager is willing to live with an unusual format for requirements and not hassle the tester about minimally sufficient compliance. The tester also appreciates that the manager may have to go to extra work to present a partially waterfall front to parts of the organization.

Quasi-Agile

A quasi-agile development environment is one where the development team is trying to apply agile development techniques within a traditionally structured organization that has policies and procedures derived from a waterfall and CMMI framework.

When this is attempted within an organization, typically one of three things happens.

1. The organization holds the line on the traditional policies and procedures, and eventually the agile teams give up fighting. The development teams retain those few agile practices that do not conflict with the organization's established policies and procedures, but the organization does not gain any substantial advantage in software development.

2. The local management agrees to support agile development as best as possible, but still has to put on a waterfall front at least some of the time. In this case the development team must practice the three principles elaborated above.

3. The organization holds the line, but the development team does not give in and the project either is cancelled or fails.

Obviously we want to avoid number three, but it unfortunately occurs too often.

Situation one can be viable but doesn't add any substantial value to the organization.

Situation two is where the action is. We will always be learning new and better ways to build software. Large organizations will always be slow to adopt them. Whether or not your company has already made the switch to successful agile or quasi-agile development, newer and even better techniques are just around the corner. So even if you've mastered the application of the three principles in this article, hang on to them. You'll need them again soon.

[1] http://www.jingproject.com/

[2] Kent Beck, Martin Fowler. Planning Extreme Programming. 2000 Addison-Wesley.

[3] Eliyahu M. Goldratt. It's Not Luck. (1994) North River Press ISBN 0-88427-115-3

Biography

Timothy Korson has had over two decades of substantial experience working on a large variety of systems developed using modern software engineering techniques. This experience includes distributed, real time, embedded systems as well as business information systems in an n-tier, client-server environment. Dr. Korson's typical involvement on a project is as a senior management consultant with additional technical responsibilities to ensure high quality, robust test and quality assurance processes and practices.

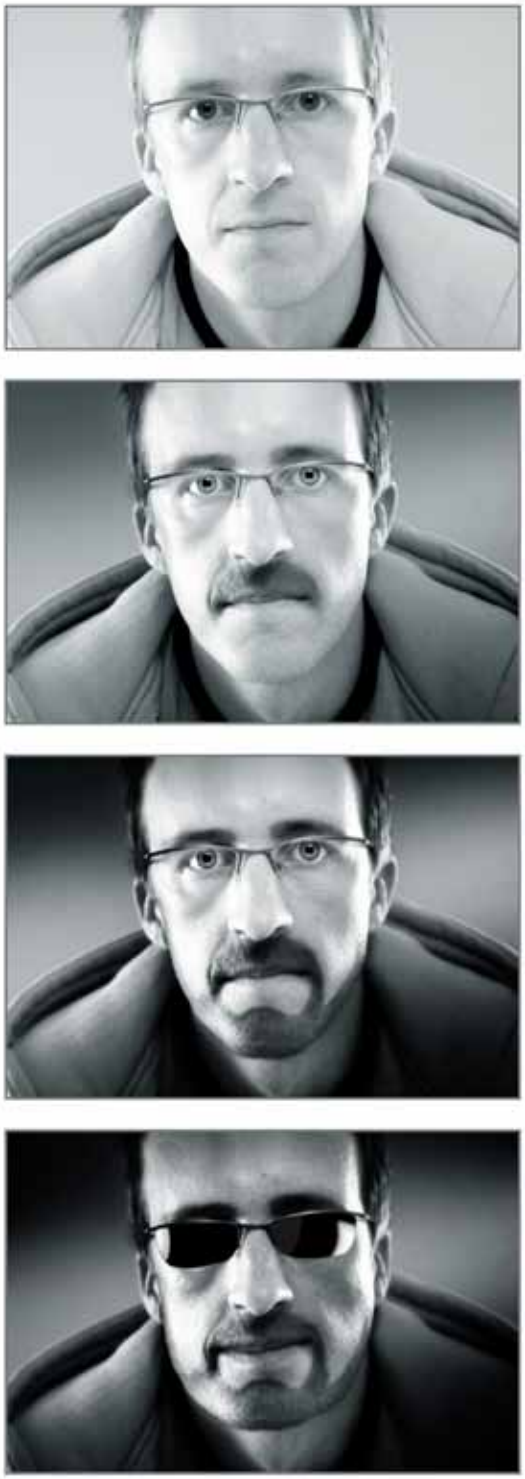
APPLICATION SECURITY

www.diazhilterscheid.com

How high do you value your and your customers' data? Do your applications reflect this value accordingly? Accidental or deliberate manipulation of Data is something you can be protected against.

Talk to us about securing your Systems. We will assist you to incorporate security issues in your IT development, starting with your system goals, the processes in your firm or professional training for your staff.

as@diazhilterscheid.com



© iStockphoto.com/abu

Co-Founder of ISSECO (International Secure Software Engineering Council)

Automated Integration Testing in Agile Environments

by Slobodanka Sersik & Dr. Gerald Schröder

The agile approach in software projects is not compatible with most of the established quality assurance processes. Quality assurance traditionally requires a finished product that must be verified against a finished specification. In agile projects, however, where response to change is more valuable than a fixed specification, a moving target must be verified against changing circumstances. Yet, testers are not able to develop test plans or automate tests more than one iteration in advance. Thus, automation of integration tests in agile environments is a difficult task. On the other hand the agile process supposes that testing happens closer to the developers in space and in time. Therefore, if the test automation effort is distributed among both developers and testers and if the test automation complexity is decreased through modularization and abstraction of reusable test components, thorough integration testing can be accomplished.

A large and growing variety of tools support automation of integration tests. Yet, most of them rely on GUI scripting by simulating users. But how can we automate integration tests for systems that

- are highly automated themselves and do not offer user interfaces?
- have use cases which are triggered by external systems?
- interact with external systems which cannot be included in the manual testing?
- offer several different user interfaces, and yet use a common back-end system?

In this article we will present a testing model we designed to specifically address these issues.

For better illustration of the problem and afterwards its solution, let's consider a simplified order and stock management system. It contains multiple front-ends: (1) an ordering user interface offered as web interface used by the consumers, and (2) a stock management desktop application used by the shipping department. Additionally, the system is dependent on an external system – the bank that actively sends bank transfers.

One typical story in the system under test (SuT) that defines a standard test case is presented in Figure 1, and can be explained through the following activities:

1. Customer places order containing goods and quantities interactively via web front-end
2. Order management system generates order reference number presented to customer
3. Customer initiates bank transfer giving order reference number
4. Bank sends bank transfers to order management system
5. Order management system matches unpaid orders against bank transfers using order reference numbers
6. For each paid order, a shipment order is being presented to the stock manager in his desktop application

How can we test this system efficiently and effectively? It can be done following a simple, nevertheless powerful model that differentiates four test development areas: (1) describing test scenarios spanning different system components, (2) implementing reusable test steps to create different test scenarios, (3) building simulators for machines or systems that cannot be integrated in the automatic integration test, (4) implementing adapters that allow a test scenario execution engine of an integration testing tool to control the different system components (or simulators).

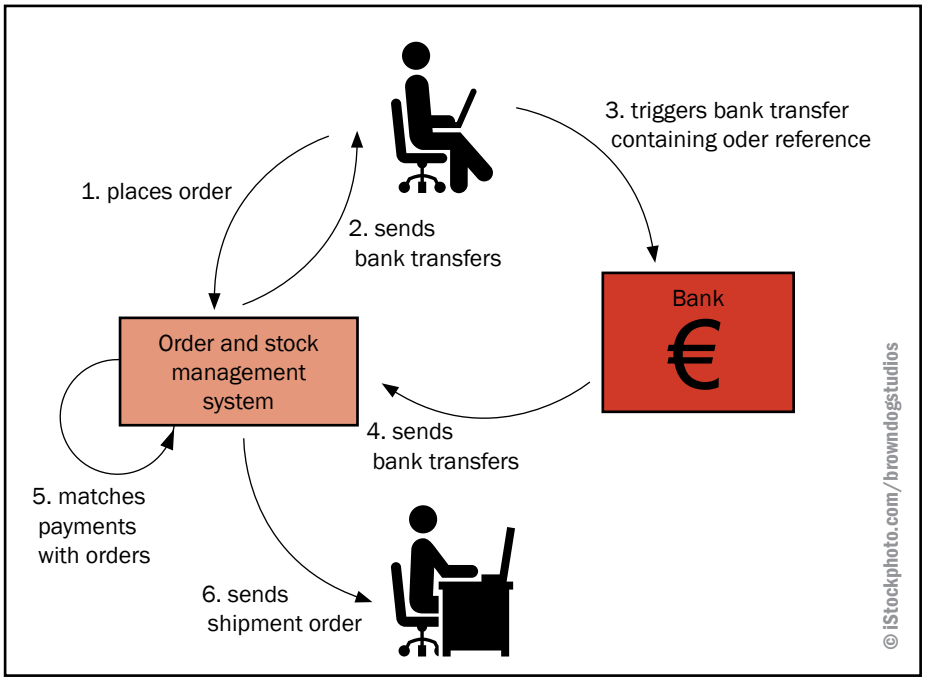


Figure 1: System under Test - Order and stock management system

Describing test scenarios spanning different system components

To distribute the testing complexity across different roles, domain testers and developers, the test description should be separated from test automation. In this way the domain testers can assemble and modify test scenarios using already available test steps implemented by the developers.

The goal of integration tests is to check the correct interaction between system components that have usually been coded by different people. Therefore testers who know and understand the overall system should prepare test scenarios spanning different system components and modularize them in test steps. The system developers themselves or test coders can implement these test steps.

In the order and stock management system described above the test scenario might be modularized as follows: place order, trigger bank transfer and read shipment order.

Implementing reusable test steps that are used to create different test scenarios

Integration tests need parameterized reusability. Why? Integration tests consist of different test scenarios that contain the same tests steps but differ in their context. For example, the placement of orders differs in the goods ordered. We do not want to implement the ordering process (list all goods, select a good from list, enter quantity, add to shopping basket, select next good, ...) for each integration test scenario that involves ordering of goods. So we create a reusable test step „place order" that is parameterized by the goods and quantities as test data. This step may be reused as a step in different test scenarios, parameterized with different test data.

Additionally, this method follows the DRY (Don't Repeat Yourself) principle that suggests a single point of maintenance. Consequently, technical changes such as new security query while ordering will be maintained in one place only, and not in each test case.

Building simulators for machines or systems that cannot be integrated in the automatic integration test

Simulators in integration testing are not just behavioral mocks (i.e., reacting to external stimuli); they have to be controlled explicitly depending on the test scenario. For example, the order system presents to the customer via its web interface an order reference number just generated to be used for payment. The integration test engine has to supply this order reference number to the bank simulator so that it may send actively (i.e. without being pulled by the order management system) a bank transfer containing this order reference number (other test scenarios may contain a distorted order reference number or a wrong sum).

Implementing adapters that allow a test scenario execution engine to control the different system components (or simulators)

Integration tests have to be robust against technical changes. We therefore advise that adapters are used in order to "wrap" the interfaces to the system under test. For example: the order management system changes its back-end interface to select goods from RMI to SOAP. Do we have to fix each integration test scenario? Hopefully not. We have abstracted away the placement of orders in an adapter used in all test steps that select goods.

The simulators are also controlled through adapters by a test execution engine. The adapters trigger active behavior and inject data into the simulators that would have been supplied by humans. Using adapters is convenient if the simulator is being replaced by another simulator or even the real system. In that case we only need to modify or exchange the adapter instead of the test steps.

Conclusion

Considering these four recommendations, the integration test for the depicted example – order and stock management system – is shown in Figure 2. Using the presented model we can build flexible and modularized tests that fully comply with the requirements of an agile project environment. We developed the model in various projects that focused on highly automated processes. Our experience showed that our approach provides an effective and cost-efficient way to build, maintain, execute and analyze automatic software tests. The introduction of this model in an agile project is a win for all four parties:

- Win for testing team: faster test automation and extreme flexibility on changes
- Win for developer team: prompt feed-

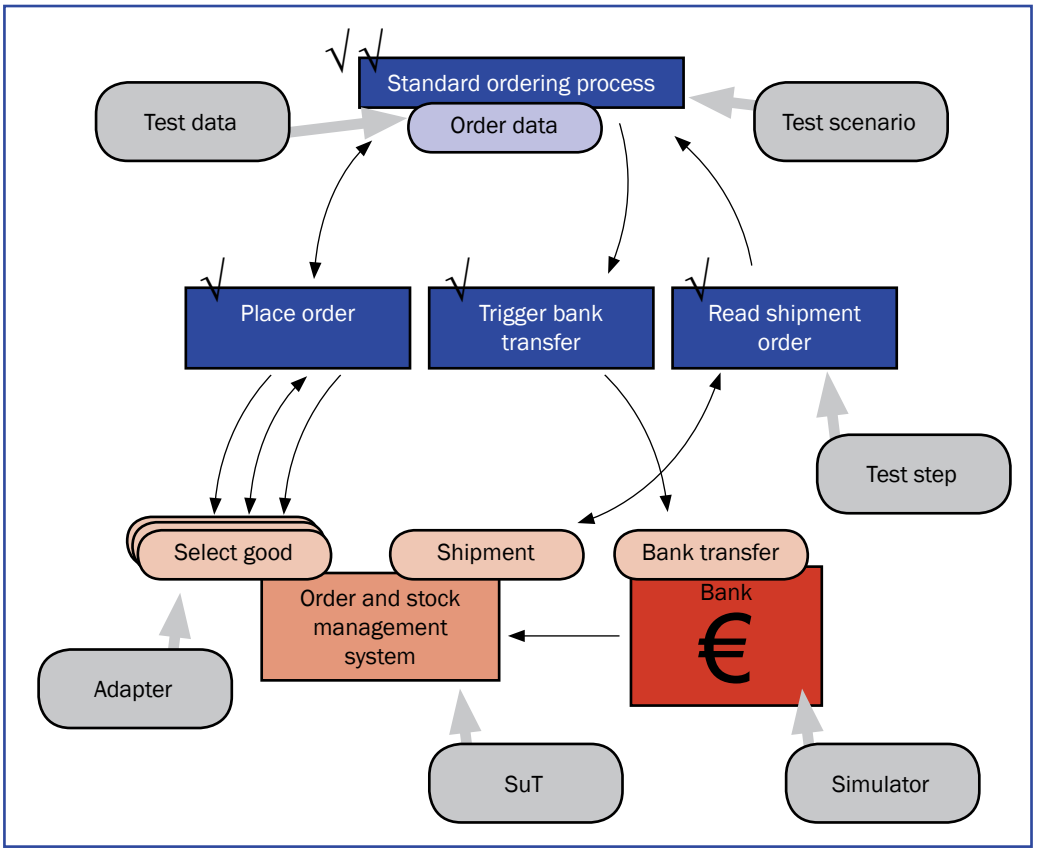


Figure 2: Integration Test - Order and stock management system

back about software quality and higher appreciation of testing efforts due to own contribution

- Win for management: lower costs due to on time failure detection and faster test automation
- Win for customers: on time delivery, high quality system

The integration testing model is implemented in an automated testing framework: the open source project iValidator (ivalidator.org).

Biography

Slobodanka Sersik is Senior Software Developer and Consultant in InfoDesign OSD GmbH. Beside her engagement on customer projects as software developer and architect, her responsibilities focus mainly on automatisation of integration and system tests. She manages currently the further development of the Open-Source Testing Framework iValidator.

Dr. Gerald Schröder is Senior Software Developer and Consultant in InfoDesign OSD GmbH. The focus of his work is put on Software-Engineering and Software-Architectures in large Java projects. Beyond that he is one of the main architects and developers of the OpenSource Testing Framework iValidator.

Training Courses by Rex Black

Managing the Testing Process

December 2-4, 2009 (Wednesday-Friday)
in Berlin

1200,- €
(plus VAT)

Pragmatic Software Testing

December 7-9, 2009 (Monday-Wednesday)
in Berlin

1200,- €
(plus VAT)

Software Test Estimation

December 10-11, 2009 (Thursday-Friday)
in Berlin

900,- €
(plus VAT)

15 % discount for two courses
20% discount for three courses

te testing experience
Knowledge Transfer

For more information and to register go to:
www.testingexperience.com/knowledge_transfer.html

Testing is a hidden project

by Miroslav Diviš

I was attracted to write this article because I read another one [1], which made me realize that "foreshadowed became obvious". I will deal with mental processes taking place in our cute brains, when we develop software or even when we test it. I focus on the informational part of these processes - information gathering and transformation, because this is true matter of software development. The management part comes later.

Preliminary thoughts

Software development is a special case of mental process called Problem Solving. I will sketch one version of this process (there is lot of such versions, I had to pick one, and this one is mine):

1. A situation occurs which we feel is inconvenient, and we decide it is time for a change.
2. We check our decision; we analyze the existing status and point out a new one (we formulate requirements which, by the way, are success criteria). If necessary we adjust our view of a given problematic situation – we extend it, narrow it down or just leave it unchanged.
3. We suggest solutions, pick one and then check this against our up-to-date knowledge and our wishes. If necessary we make corrections of any of our previous conclusions and suggestions.
4. We implement the chosen solution, if there is one. During the implementation we gather new information, we compare them with older ones and eventually we amend our practices.
5. We test the implemented solution against our requirements, and usually we have to make corrections – as in every previous step.

There are many such descriptions; some have more or fewer items, some give names to items, but I hope we understand each other. We discover two detail levels in the description: in the higher one we see separated items called phases, stages (I defined just five such phases above). Their order is fixed and every phase is ended explicitly; some artifact is produced (document, application). But we have to distinguish a more detailed level, where each phase is a process interacting with any other phase/process – forwards and backwards. Perhaps such a phase should be called an "open" phase, because it has to accept all changes raised by the problem elaboration process. These properties of software development were recognized and incorporated into some development methodologies, I mention at least AUP in [2].

If the problem is solved in a team, the work has to be divided between team members and this situation leads to specialization. Experts have to communicate and coordinate their work. So we need an interface – better two interfaces: one for information exchange, and one for managing. Both interfaces are important, and they must not be interchanged. As I mentioned above, this article is about the information interface.

A set of artifacts is issued and published at the end of each solving phase. The set must be true, consistent (containing no contradictions) and comprehensive. I will call such a set a system. According to the problem solving approach just described we may say that software development is a step-wise transformation of artifact systems which is getting closer and closer to the desired solution (being the solution fixed or not). The main property of transformation is this: the result of every interleaving phase must be a system. If I add, delete or change any artifact, the three system properties must be satisfied. This is ensured by a permanent checking of the properties and by making corrections. If some new information is inappropriate, I change it – or I change the older version of system! So this is the way life goes on: either fit into the system or change the system itself.

What testers do crops up twice in my problem solving description mentioned earlier. The first manifestation is explicit – I used word "to test" in the fifth phase above. This is to be expected because we are generally used to doing testing. The second manifestation is implicit – testers check three system properties every time one phase of software development passes to the next one. Actually, true software testing is nothing different than checking that another artifact, i.e. installed application, is in conformance with all other artifacts. The same approach is proposed in the W-Model!

Conclusions

Testing software is hypotheses testing

I would like to start with one self-evident banality: the testing (not only of software) is a comparison of two states – expected (or required in case of software development) and achieved. A few words for the second state, because it is easy for testers to get it. Someone just installs the last version of the tested software - and it is done. I suppose you never met a manager who asked you to test before the installation. The situation in the first state is absolutely different. It is very hard to get this state (it arises from the transformation of many input documents – which may be inaccurate, incomplete,or generally of a poor quality) and while a lot of information is missing, you are asked to finish the testing!

In testing, the Test Condition acts as a hypothesis. The scientists never say "hypothesis was confirmed", they say "hypothesis was not disconfirmed". So do we testers – tested software is never without errors, we were only unable to find some within a given time.

Testing is only one

Testing has only one problem domain. It is independent of methodologies and tools used in other phases of the software development life cycle, and it has its own unique and uniform rules and properties.

The testing domain is formulation, administration and the testing of hypotheses. I am not sure the following example will be clear enough, my life was more colorful than the life of an ordinary IT man, but I have no better: the relationship between testing and other parties of SDLC is the same as relationship between bookkeeping and business. Whatever your business branch is, whatever the size or the form of you company, you are obliged to do the accounting. Accounting has its own laws and standards, tools, best practice, education, certification. The business is creative and the accounting is rigid. Accounting is accurate and incorruptible. All this is true for software development too.

Independence of testing on methodologies and tools used in other phases of SDLC seems to be apparent. There is only one interface among us and the rest of the world – the handing over of documents. Whatever the world does, it must hand over documents to us testers at the right moment.

The unique and uniform rules and properties of testing are:

- Repeatability and ability to substantiate
- Absolute dependence on intermediate information
- Its own complexity
- Its own methodologies

Testing inherited repeatability and the ability to substantiate from its scientific parent. For these purposes we make our own documentation and we also file progress and results of testing – practically the same information as we use for the description of detected errors.

The tester never meets problem domain information directly; he/she works only with information produced by the software development team. This is why testers ask for inputs to be exact, comprehensive, in time and reasonably presented. With the exception of extremely small, short and isolated projects, there is only one acceptable way of information sharing – fixed form on an external carrier, i.e. documentation. Do not forget the most expensive part of software life - the maintenance!

I leave testing complexity, methodologies and early detection principle without further comments for now.

Testing is a hidden parallel project

Actually, this is obvious now, isn't it? If not, see project definition from [3]: the project is a temporary endeavor with a beginning and an end, creating a unique product, service or result, which is progressively elaborated. And why is it hidden? Because nobody treats testing as a project! Just for fun – testing apparently has its own test phase. If you want to dispute that all phases of SDLC might be projects, please do not forget their time limits - they are only subprojects. Testing is omnipresent, working for the same duration as the main project and affecting fundamentally the whole main project. The same applies to the accounting for a business. Long live testing, the parallel project!

Finale

I prefer to know the fundamentals of any job I do. I like to find out a unity in our world, broken up by specialists into senseless details.

So you testers, be proud of your work and require always a fair portion of respect – to your position and to true naturalness of your job.

References

[1] Christie, James: The Seductive and Dangerous V-Model in Testing Experience 4/08

[2] Ambler, Scott. W.: his www. Ambysoft.com pages

[3] PMBOK Guide – Third Edition

My great thanks go to creators and contributors of Wikipedia, a really exiting source of knowledge.

Biography

Miroslav Divis, mathematician by education, man of many crafts - software developer, civil engineering manager, carpenter,... For the last three years engaged in software testing, Prague Accenture ex-Testing Capability Lead, now freelancer.

How to start your journey into automated testing?

'Taste of Python for testers'

by Péter Vajda

Motives

I often meet people coming from manual testing teams with great sense and experience in their profession but lack of test automation knowledge. How should they start a journey into automation and learn more about software development in the age of agile? Maybe to pick up a great scripting language, some automation techniques and then start to apply them in work situations. My suggestion is to write your first application and tests to challenge your brand new code at the same time. That works much better than learning only fundamentals for weeks. You will learn them on the way anyway! If you are not so familiar with the common programming terms like class, function, variable, conditions and loop, you can use Python's excellent documentation [1] and the great Dive Into Python book [2] to understand the code snippets. Don't worry, Python is easy to understand and pick up. I also try to hit many birds with one stone by giving a taste of Python and present a couple of useful practices: TDD (Test Driven Development) and data-driven testing.

So what should our first simple application be? Usually, quick coding tasks are given for test engineer candidates in job interviews: for example generate the Fibonacci number for index n and test the implementation with automated tests.

Start testing with Python

What is the first, most trivial test case?

```
>>> fibonacci(0)
0
```

Calling fibonacci(0) should return the number 0 . The example above is using Python's interpreter, which behaves like a Unix shell. How can this test be formalized? Luckily we have "unittest", Python's unit testing framework, available:

```
import unittest

def fibonacci(n): pass

class TestFibonacci(unittest.TestCase):
 def test_0(self):
 self.assertEqual(fibonacci(0), 0)

unittest.main()
```

We start with a placeholder Fibonacci function just to be able to call it. Then TestFibonacci class contains our first test: test_0. Checkpoint assertEqual verifies fibonacci(0) returns with 0. So let's run our first test:

```
$ python fibonacci.py
F
======================================
FAIL: test_0 (__main__.TestFibonacci)
--------------------------------------
Traceback (most recent call last):
 File "fibonacci.py", line 7, in test_0
 self.assertEqual(fibonacci(0), 0)
AssertionError: None != 0

--------------------------------------
Ran 1 test in 0.000s

FAILED (failures=1)
```

It fails of course, but it is time to change the Fibonacci function to pass our first test:

```
def fibonacci(n):
 return 0

$ python fibonacci.py
.
--------------------------------
Ran 1 test in 0.000s

OK
```

Do not worry about the code if it seems weird; we will soon improve the design. However, we have only one test so far.

Evolution of the design

So let's make our second test to be able to write something better in our Fibonacci function:

```
def test_1(self):
 self.assertEqual(fibonacci(1), 1)

$ python fibonacci.py
.F
======================================================
FAIL: test_1 (__main__.TestFibonacci)
------------------------------------------------------
Traceback (most recent call last):
 File "fibonacci.py", line 11, in test_1
 self.assertEqual(fibonacci(1), 1)
AssertionError: 0 != 1

------------------------------------------------------
Ran 2 tests in 0.000s

FAILED (failures=1)
```

How to make it pass? Maybe like this:

```
def fibonacci(n):
 return n

$ python fibonacci.py
..
------------------------------------------------------
Ran 2 tests in 0.000s

OK
```

Great, but how can we move on? Too many test functions and lots of code to write, so let's drive our tests with data instead."

Test against reference

If we look for a Fibonacci reference then data can be re-used and we do not need test_0 and test_1 anymore but a new test against the reference:

```
reference = [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55,
89, 144, 233, 377, 610, 987]
def test_fibonacci_against_reference(self):
 for r in self.reference:
 self.assertEqual(r, fibonacci(self.reference.
index(r)))
```

Use recursive calls of fibonacci to get all the tests pass:

```
def fibonacci(n):
 if n <= 1 : return n
 else: return fibonacci(n-2) + fibonacci(n-1)
```

Error cases

What should happen if we call fibonacci with negative numbers or non-integer parameters? (Let's not use the Fibonacci definition for negative index in this example): Throw the corresponding exceptions. Test to check on negative parameters:

```
def test_fibonacci_raises_IndexError(self):
 self.assertRaises(IndexError, fibonacci, -1)
```

And let's iterate through some non-integer types as parameter:

```
def test_fibonacci_raises_TypeError(self):
 for i in ("", [], (), 1.1):
 self.assertRaises(TypeError, fibonacci, i)
```

New version of Fibonacci function, which throws exceptions when incorrect parameter is used:

```
def fibonacci(n):
 if type(n) != int: raise TypeError
 elif n < 0: raise IndexError
 elif n <= 1 : return n
 else: return fibonacci(n-2) + fibonacci(n-1)
```

Final Source Code

```
import unittest

def fibonacci(n):
 if type(n) != int: raise TypeError
 elif n < 0: raise IndexError
 elif n <= 1 : return n
 else: return fibonacci(n-2) + fibonacci(n-1)

class TestFibonacci(unittest.TestCase):
 reference = [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55,
89, 144, 233, 377, 610, 987]
 def test_fibonacci_against_reference(self):
 for r in self.reference:
 self.assertEqual(r, fibonacci(self.reference.
index(r)))

 def test_fibonacci_raises_IndexError(self):
 self.assertRaises(IndexError, fibonacci, -1)

 def test_fibonacci_raises_TypeError(self):
 for i in ("", [], (), 1.1):
 self.assertRaises(TypeError, fibonacci, i)

unittest.main()
```

And all tests pass, so we are done:

```
$ python fibonacci.py
...
------------------------------------------------------
Ran 3 tests in 0.006s

OK
```

Conclusion

We used the TDD cycle during this article: write a failing test -> make test pass -> refactor. Yes and you can even refactor tests to have better verification. TDD is a very powerful design technique and the side-effect is plenty of automated tests, which can be run in a continuous integration system to make sure all integrated code works.

Where should we go from here? Python is easy to master and can be used from the command line testing to Web and GUI testing. Unit testing framework is an easy way to run and formulate even higher-level test cases. You may use the references section to learn more about Python, unit testing framework and TDD. Of course all these techniques can be applied in other higher-level programming languages as well.

References

[1] Documentation for Python http://docs.python.org/

[2] Dive Into Python http://diveintopython.org/

[3] Growing Object-Oriented Software, Guided by Tests http://www.mockobjects.com/book/

Biography

Péter Vajda gained MSc in Teacher of Physics and BSc in Teacher of Mathematics at the University of Pécs, Hungary. Later he obtained the ISEB Practitioner Certificate in Software Testing. Péter has been involved in software development, test automation, test analysis and test management of Web systems and databases for the past 9 years. Recently he is working on embedded Linux development and coaching agile teams for testing practices such as test- driven development, continuous integration and behavior-driven development.

TAKING THE STRESS OUT OF TEST AUTOMATION

Experience leading edge test automation technology

expecco

- model-based test development
- let it run
- cost-effective
- time-saving

For more information, visit our website:
www.exept.de

eXept SOFTWARE AG


© iStockphoto.com/hidesy

Agile Collocation

Combining the powers of Agile Testing and Global Delivery Model

by Uday Ghare & Ravi Sheshadri

As Charles Dickens once said, "It is not the strongest of the species that survives, or the most intelligent, it is the one most adaptable to change." This has never been truer than in the current recessionary environment, and is thus an important part of successful post-recessionary strategies. To survive in the market place, the customer has to adapt to change rapidly and on reduced costs.

The Agile model is based on the concept of adaptive methodology. The core differentiation from the traditional waterfall model in the Software Development Life Cycle [SDLC] is that there is iterative agile testing, early participation of all stakeholders (including the customer), early visibilty of the product and an iterative approach towards building the final product. In a nutshell, this is an approach that adapts to change in its entirety.

While the advantages of agile testing are obvious and well known, it is imperative to understand its challenges/disadvantages such as high costs, heavy onshore and people-dependent communication. This paper aims to understand and address the challenges of agile testing by introducing an offshore collocation center. This collocation center works towards making the agile testing process quicker, faster and, most of all, cheaper by amalgamating the agile process onto an effective offshore-based delivery model. However, before moving onto Agile collocation center, it's of utmost importance to understand and appreciate the innovative way of offshoring (i.e. the global delivery model).

Global Delivery Models (GDM) have been instrumental in providing cost-effectiveness on IT spends for customers around the globe. The access to low-cost and efficient resources with heterogeneous skill sets and core domain competency have been the drivers for GDM. The advances in IT computing and communications have made this possible and have served as the impetus to encourage customers to look beyond geographical boundaries in order to gain cost advantages. GDM has been successfully implemented for large-scale programs on varied SDLC and has proven to reduce costs and increase access to a larger pool of skilled experts.

So how does one implement an Agile Testing strategy in an offshore collocation model?

Agile Testing demands continuous integration testing across iterations and test phases. Throughout the phases there are two keys aspects:

1. Effective communication channels for faster issue resolution, and
2. Test environment availability and resilience to accept continuous code releases.

GDM also necessitates better communication channels, as the overall work is scattered across time zones, locations and sometimes different vendors. Therefore, communication is a common success factor in both Agile and GDM, and so we introduce the model of "Agile Collocation".

Agile Collocation is an approach to get all key stakeholders on the program together in a single location, with higher representation from the testing community, preferably 60 to 70%. The remaining 30 to 40% would be representatives from component development team, solution design team, environment support, business representation etc. The greater percentage of testers is necessary to maintain high levels of throughput in the testing phases. Also, this is imperative, because success on an Agile testing project is most often derived by test-driven development and delivery management.

The choice of the location could be based on where the maximum cost benefits are derived. For example, if the customer has a lot of work distributed in an offshore center, then an Agile Collocation center could be built offshore at one of the vendor premises through an RFP process or mutual agreement.

Another important aspect to further expedite testing is that the Agile Collocation center's network infrastructure can be setup as a customer's extended network. The advantage is that the people working in the center will be part of the customer network with:

a. no access to their individual company networks/intranet resulting in data security, and

b. fewer firewall restrictions resulting in lower transaction response times

Agile Collocation is a continuously running center in which the testing team will have direct access to representatives from business, project executives, solution/process design and individual component/application development representatives. With direct coordination, the model tends to expedite the defect resolution process and also ensures direct ownership for all issues.

Key factors responsible for the success of an Agile Collocation center include:

1. Clear and continuous communication
2. Trust-based relationships
3. The ability to acknowledge and overcome cultural challenges
4. Building remote domain knowledge
5. Technical infrastructure to support distribution

Key features of Agile Collocation are:

- **Test-driven development**

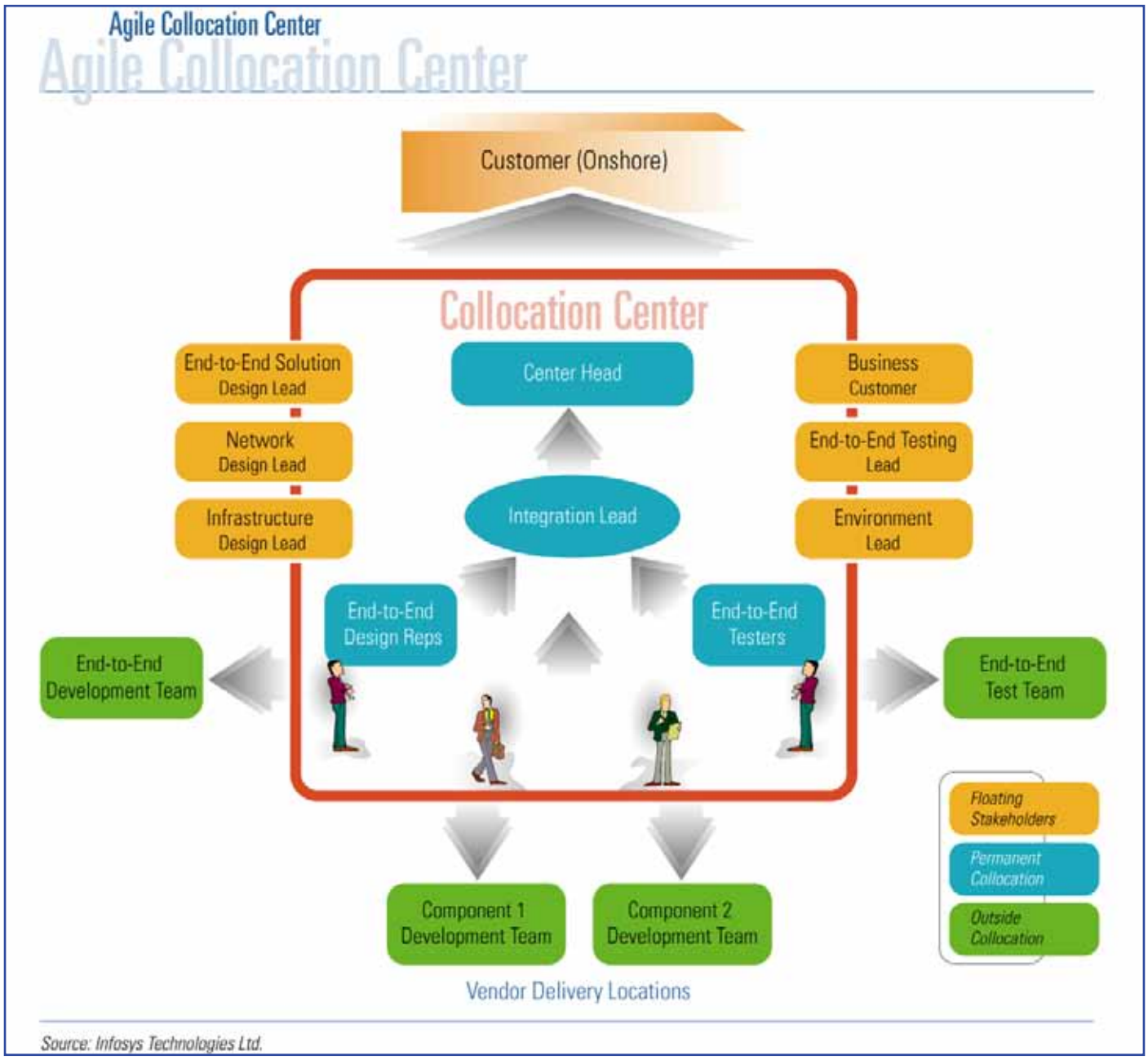
- Test leads driving the delivery by chairing the scrum/daily calls with decisions made on daily code drops, environment/model changes and defect management.

- **Ownership of issues**
  - Improved process resulting in faster turnaround time
- **Better communication**
  - No emails, emphasis on face-to-face communication resulting in faster issue/defect resolution.
- **Better end-to-end coordination**
  - Design, development, test, business and across E2E delivery management under one single roof, with one single common objective of delivery excellence.
- **Increased senior management focus**
  - Emphasis on problem resolution, forward planning and effective escalation process for blockers.
- **Coordinated collocation model**
  - Model cutting across the program resulting in better information sharing, in a regular cohesive fashion.

Customers receive enormous cost benefits and effort-savings due to a test-driven Agile Collocation model. By transferring more work to the Agile Collocation center, the need for maintaining people onshore is reduced considerably, and the customer could go with a reduced onshore team and the remaining people at an offshore collocation. While onshore costs are reduced dramatically, Agile Collocation also provides the added benefit of direct offshore effort reduction due to cutting down on communication delays, continuous environment monitoring and faster, quicker resolution of defects and show stopper issues. A typical offshore effort saving of 40% is envisaged by customers implementing an Agile Collocation Center.

A well implemented Agile Collocation Center brings in huge cost benefits due to a high degree of offshoring, optimization of efforts by effective coordination and direct ownership of defects/issues. Agile Collocation helps to reduce time-to-market and total cost of ownership. More importantly, this model brings the best out of all stakeholders, to achieve the program commitments, by establishing a clear vision, continuous communication and trust-based relationship.

It is vital to understand that this model inherits all the values and practices of the agile process, and the model is to be used in conjunction with the agile process, not instead of it. Agile excellence by offshoring is an innovative extension of the agile process and not a replacement for the agile process.



Biography

Uday Ghare (Uday_Ghare@infosys.com) is a Group Test Manager with Infosys Technologies. He has over 13 years of IT experience in the Telecom and Media domains.

Ravi Sheshadri (Ravi_Seshadhri@infosys.com) is a Group Test Manager with Infosys Technologies. He has over 13 years IT experience across Banking, Energy & Utility, Communication, Media and Entertainment domains.

ISSECO®

Certified Professional for Secure Software Engineering

TRAINING

There is no doubt that application security is a vital issue for any modern application. Unfortunately there is no plug that you can press to turn it on. Application security should be implemented throughout the product life-cycle.

The ISSECO course will teach you the how and when. The course will introduce the technology but never the less the methodology. Furthermore you will learn the security logic and how it can be achieved without breaking the budget.

Contents

- View Of The Attacker
- Explain The Definition Of The Terms Hacker, Cracker And Ethical Hacker
- Show Examples Of Famous Hackers And Their Attacks
- Point Out The Different Skill Levels
- Modes Of Hacking
- Hacker Motive
- Illustrate The Process Of Hacking
- Outline Common Hacking Tools
- View Of The Customer
- Explain Why Customers Expect Secure Software
- Trust & Threat Models
- Methodologies
- Requirements Engineering
- Secure Design, Secure Coding, Secure Testing, Secure Deployment
- Hands-On Workshop
- Security Response
- Explain The Difficulties Of Fixing Security Issues Via Standard Maintenance Processes
- Code & Resource Protection

Tutor: Manu Cohen-Yashar

02.12.09 - 04.12.09 **in Frankfurt am Main, Germany** **Price: 1899,00 € (plus VAT)**

Register at http://training.diazhilterscheid.com

ISSECO®
SECURE SOFTWARE ENGINEERING

SELA GROUP

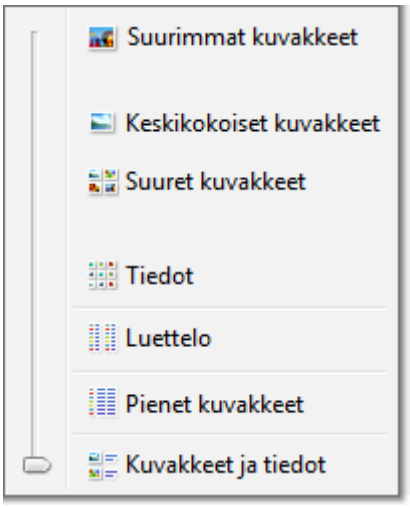
Díaz Hilterscheid

The Importance of Language

by Michal Kolář

Introduction

Take a look at the following screenshot and answer the question: "What is wrong with this well-known menu?"



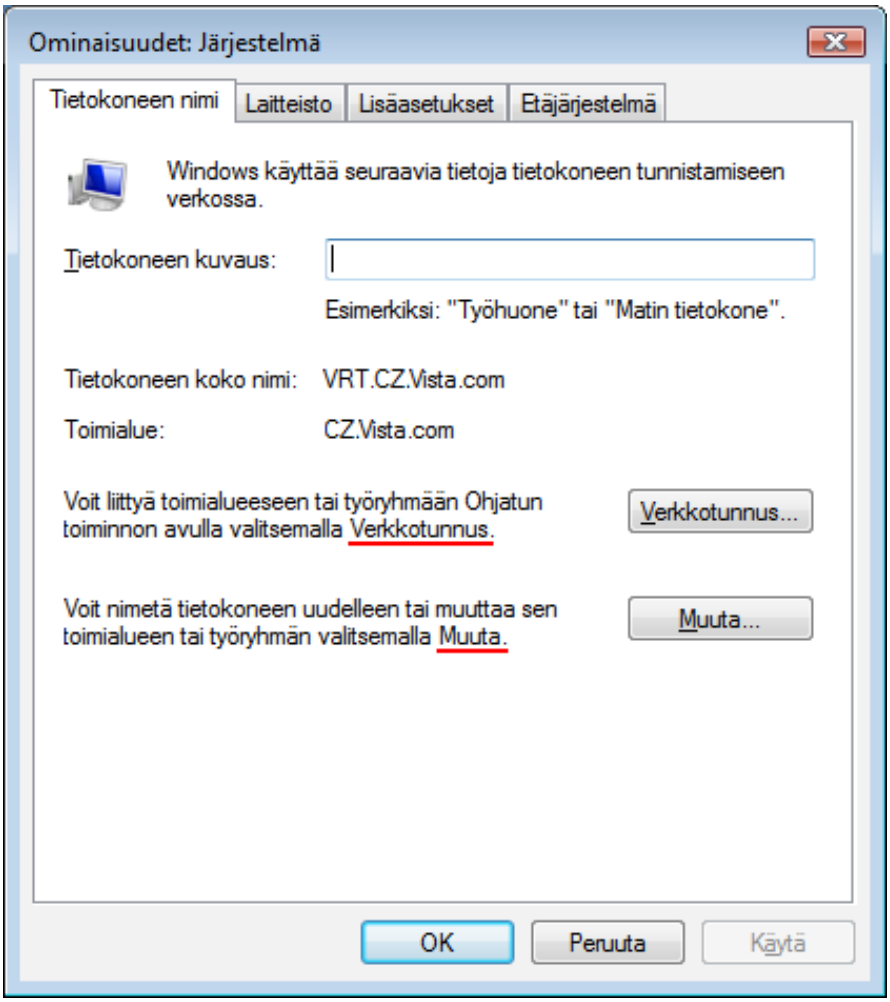
Do you think perhaps that menu items two and three (*Keskikokoiset kuvakkeet* meaning "Medium icons" and *Suuret kuvakkeet* meaning "Large icons") are not aligned well and have too much space around them? Well, believe it or not, these layouts are not wrong, or at least this layout truly reflects the original English-language layout used in Vista. Give it another try. Still nothing?

Unless you were born in the Land of a Thousand Lakes (or learned Finnish somewhere along the way), you have no chance to discover the problem. The problem is that some text labels are transposed – their text is swapped with other labels while the icons remain in their original positions. So, the text for "Large icons" is next to the medium-sized icons, and the text for "Medium icons" is next to the large-sized icons. There is almost nothing that regular functinal or localization testing can do to detect such an error, but – as always – there is a way to deal with this kind of problem: **linguistic testing**.

You might be a world-renowned producer of a large software application which is localized in twenty languages, or you might be a stand-alone developer with a great small tool and you are thinking of localizing your application to break into global markets – in both cases linguistic testing is something you should definitely be aware of. It's not enough to have your translated resource files go through a QA process by another linguist (though, having translated resource files reviewed by someone who didn't work on localizing in the first place is a good idea). Many UI elements are displayed only in the live compiled software and may actually require some significant infrastructure to be running before they appear. You need to see these live and in context, just as the menu displayed above.

What linguistic testing is and what it is not

Linguistic testing is sometimes confused with localization testing. You can visit Wikipedia and search for definitions there, but let me offer really simplified definitions of each:



- **Localization testing** focuses on the correct functionality, appearance and completeness of the localized product.
- **Linguistic testing** takes care of ensuring the correct language rules are being used and focuses on correct in-context linguistic usage.

The border between these two types of testing can be a bit blurred and there are several cases where a localization test engineer can easily report a problem that would normally be expected to come from a linguistic reviewer. And vice versa.

In this typical dialog box, the button names the user is expected to click on if the described action must be taken are noted (here, *Verkkotunnus* meaning "Domain" and *Muuta* meaning "Change"). But the typical software localization process may not be straightforward. The individual UI elements may come from different resource files; some may come localized by another translator, some may be recycled from a previous version of the same product and "locked" for any changes. That's why verification of the consistency with the real button label is something which is expected from both a localization test engineer and a linguistic reviewer. An inconsistency within UI (such as button label conflicting with information in text) may be introduced easily in the localization process, and in practice occurs much more often than one may think. The value of a linguistic reviewer in such cases is much higher for languages that are perhaps too "exotic" for the localization test engineer to be expected to notice a possible error: for example Korean for a European tester or Hebrew for a Chinese test engineer.

Depth of linguistic testing

There is always a dilemma that must be decided at the very beginning of a software localization project – you need to know what level of detail is important to you and where to draw the line of not needing to take care of every single detail. To ensure linguistic quality, there are basically two different approaches that affect how testing will be completed, the target quality, and of course the cost. Those two approaches are:

A. **In-context review** – With this approach, the reviewer just skims the dialogs boxes, menus and pages and verifies that the localized content corresponds to the expectations of a user. For example, in an e-Learning course for children, there might be a picture of a saucepan; the reviewer verifies that the content next to the picture is really about cooking and not, for example, about African frogs.

B. **Linguistic analysis** – In this approach, a professional linguist goes through all the strings displayed on the screen and carefully observes all language aspects and verifies that the linguistic conventions regarding consistent and appropriate use of language, proper use of terminology, correct style and tone are strictly followed, and that the localized text corresponds to the correct functionality of the UI.

To decide which approach is best for a given project, we need to think of the real users. Who they are? If the software application is an automotive diagnostics analysis tool, then it will most probably be used by people who do not care about every single language aspect followed to the letter and therefore it might be appropriate to choose an in-context review approach. This type of review can be a more cost-effective approach compared to a more thorough linguistic analysis, which should always be considered when testing educational software, retail applications, etc.

Each approach requires different testers. As outlined above, the in-context review can be more cost-effective, because this type of testing does not necessarily have to be done by linguistic professionals or by native speakers of the target language. It is a perfect job for part-time employees, company employees who study the language in question or have experience with the foreign country (through marriage, work experience, etc.), or, even students from local universities who study and like the language.

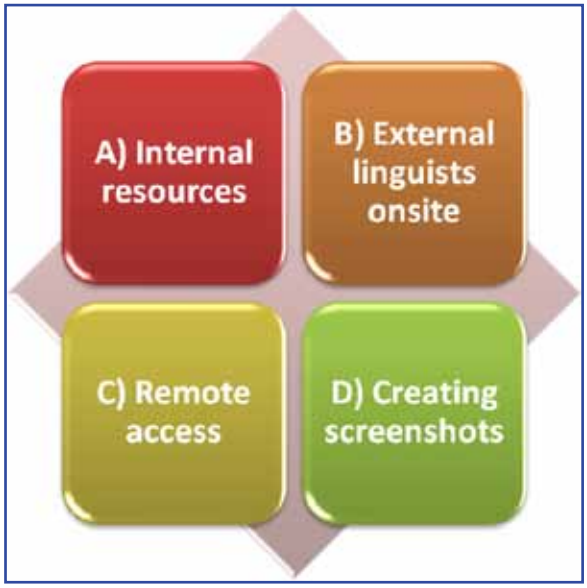
On the other hand, for the more thorough linguistic analysis you usually need "the best you can get." You might think, "Great, I'm in luck here" if the language that you would like to do linguistic testing on is your mother tongue (or someone on your team's), but hold on. Are you absolutely sure that you understand every single aspect of the language? Are you really enough of a linguistic expert? Personally, I was pretty good in school with languages, but now as a computer testing guy I would be very afraid of using the remnants of ten-year-old knowledge to test the linguistic quality of a product – even in my native language.

In most cases, though, a native speaker is wholly appropriate for linguistic analysis. Only in very special cases (such as testing software for, let's say, post offices or any other public institution), where 100% correctness is an absolute must, a native speaker with a linguistic degree must be found. But, be ready for the fact that such people have considerably different expectations to students when it comes to how they expect to be compensated for their work.

It is often the case that there are limited resources for less widely-spoken languages and there might be a situation where you might have to consider using the same person who translated the product to do the linguistic testing as well. This does not necessarily have to be a problem, and for an in-context review this solution is fine. However, for detailed linguistic analysis it is definitely not suitable, and, the added value of such testing with the same resource is very poor.

Picking the right approach

There are some "best practices" that can be followed in order to perform linguistic testing. No matter if you do it yourself or if you ask a vendor to do it for you, there are basically four approaches that can be considered when it comes to linguistic testing:



A. **There are internal resources within the company** – This is probably the most comfortable solution, but it may not necessarily be the cheapest one. Maybe your senior developer can speak Hindi and would be willing to help out with testing. However for in-context review you would be just fine with a student who would be happy to take on the task at a cost of, say, 30 per cent of your senior developer's cost. The student – even with some management costs – might be the more cost-effective solution.

There can also be a situation where you need to perform linguistic testing on ten languages in total, with only one language covered internally by your employees. Think twice about whether it is good to split the work and test each language in a different way. The overall efficiency may suffer and you may find out in the end that utilization of one of your internal employees did not actually pay off.

B. **External linguists brought onsite** – Probably the most reliable solution and one where you have full control over the testing process is to have linguistic resources come onsite. With onsite linguistic testers, you can adjust project

details as soon as issues appear and you can also minimize downtimes by taking advantage of proximity to resolve any problems as they are noted. However, this type of testing is extremely time-intensive in the planning phase and requires a rock solid project plan that predicts in advance that the work will be ready for testing on a particular date and no later (because you need some time to contract all those onsite linguists.)

For example, if you need to coordinate ten external linguists for ten languages to perform five-day linguistic testing onsite at your premises, you should definitely have at least three different testing timeslots accounted for in your planning, and try to allocate groups of linguists in those timeslots. It is impossible to agree on a single testing timeslot that would suit all the linguists as well as your production team.

C. **Setting up remote access** – Sometimes it is not possible to bring linguists onsite. The nature of your product might mean you don't need "top secret" approaches, and it's acceptable to allow for remote access. Or, you may simply be just out of office space. A solution is to set up the right infrastructure and grant remote access to your linguists.

This is a good approach for projects where you need to test very specific languages or you need to test a large number of languages and there is no chance to bring linguists onsite. It also gives you more flexibility, as remotely-connected linguists are more willing to accept some changes in the schedule should tasks slip. With this approach, you miss the benefits of direct support and you must rely more heavily on the linguists to do exactly what they should. Sometimes this approach cannot be applied because of technical reasons (too complex infrastructure, target device does not support remote connection, etc.).

D. **Creating screenshots** – This final method is very suitable for technically demanding products where it is not possible to bring all linguists onsite. You can capture screenshots of all the screens you want to test and send them to the linguistic testers. The screenshot approach still provides some context to the testers, which is sufficient for full understanding. It is also possible to create a visual screenshot map so that the linguists can clearly see what actions must be taken to get to that particular dialog/menu/page and therefore increase awareness of the context for the linguists. Screenshooting is always a quite time consuming activity, but often can be automated by using automation tools and preparing a script which is later run on all language versions, generating screenshots in a few minutes (or hours at most). Here, the more languages you are testing, the better. Preparing automation scripts for one or two languages may not pay off, but for ten or more languages the situation is certainly different.

Preparation of screenshots requires more work from the technical team but eliminates confusion with more complex products where problems can be expected if the product is tested using remote access. However, for projects with a very large number of screens that need to be checked (200+) it may not be the perfect solution, as the number of screenshots may cause confusion for the linguist and special attention must be paid to the organization of the project.

What else needs to be done?

No matter what method is chosen, and no matter what quality level you targeted, it is always essential that the reviewer understands the whole concept of the tested product; that he/she knows what it is used for; what are the key features; what has changed since the last release; who is the target user of the product and any other things that may help him/her to understand the background. This level of detailed understanding is very important for the reviewer to be able to verify the context consistencies, and it is vital for the linguistic testing approaches where reviewers are not onsite, or when they are working outside of your normal business hours.

Sometimes there is a push to combine the roles of linguistic and localization testers and have everything done by a single person at the same time. Tempting, but difficult to fulfill. Linguistic reviewers are usually liberally-educated, while localization engineers are typically more technically orientated. There is a chance of combining localization testing and the more simple in-context review, but as mentioned above, the more languages that are to be tested, the more complicated this approach is.

Another important thing is to clarify workflows and set firm rules before localized product testing is started. A list of detailed instructions must be prepared so that the reviewers can clearly see what they should verify on each dialog box, menu, etc. , and just as important, what they should not bother with. A number of organizational things must be clarified as well: Where to log defects? How to report the progress? Who is the contact person? Another aspect is to define the process to follow later when the defects are fixed and verifying that the fix has made it to the final product. Is it still necessary to have a language specialist review the fixed element again, or is it enough that the defect really seems to be gone?

Conclusion

Linguistic testing of a localized product may not be an easy undertaking. There are several aspects that complicate matters and it is often inefficient for development companies to "do it on their own."

There is no "best way" that can be recommended from the scenarios outlined above. Each has some advantages; each has some disadvantages. The most suitable solution needs to be selected based on the project type, schedule, budget and company preferences.

There are many localization companies that offer this type of service and are able to deliver everything as a single package, and this is probably the future of linguistic testing – it is something that is performed by specialized companies with a strong tradition in providing linguistic services. This is the right approach for anyone who cares about quality. Increasingly, more and more organizations, large and small, recognize that the model "we'll give it to some of our local representatives" does not meet their needs and therefore they seek professional localization companies that deliver professional quality at a reasonable price.

Biography

Michal Kolář, Test Manager at Moravia Worldwide's Testing and Engineering business unit, joined the company in 2000. Michal started off with the company providing localization testing quality assurance for Moravia's key software clients, later assuming responsibility for testing quality control processes. In his current role, Michal is manager of a testing unit at Moravia headquarters in Brno, Czech Republic, where he maps and monitors production processes applying Event-Driven Process Chain (EPC) methodologies. He has also initiated the implementation of new technologies such as virtualization. During his career, Michal has prepared numerous testing and QA training courses and has trained hundreds of test engineers. He is fluent in Czech and English, with a working knowledge of Russian.

ISTQB® Certified Tester Training in France

www.testplanet.fr

a Díaz & Hilterscheid partner company

Testing Services

www.puretesting.com

PureTesting
Testing Thought Leadership

Díaz Hilterscheid

IREB

Certified Professional for Requirements Engineering - Foundation Level

http://training.diazhilterscheid.com/
training@diazhilterscheid.com

16.09.09-18.09.09 Berlin
18.11.09-20.11.09 Berlin

Training by

SELA GROUP

www.sela.co.il/en

Improve your test skills!

RSI

RSI Training Center

ISTQB Accredited Training Provider
Rex Black exclusive partner in Brazil
Basic to advanced test trainings
Open and in-company classes

contact: ctr@rsinet.com.br

http://portal.rsinet.com.br/ctr/

ISTQB® Certified Tester Training in Spain

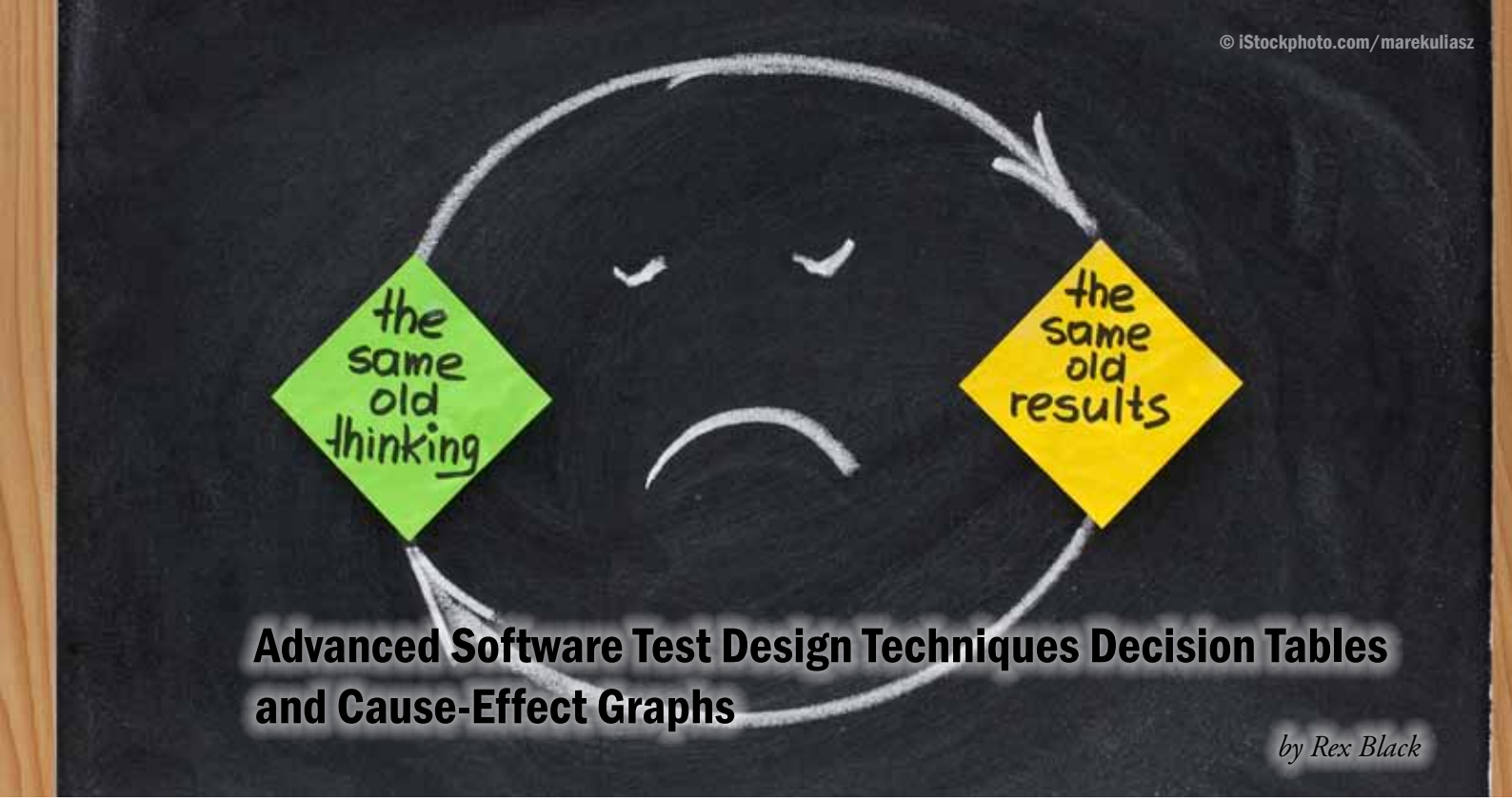
www.certified-tester.es

ISTQB® Certified Tester Foundation Level

for only 599,- €
incl. exam & plus VAT

ONLINE TRAINING
English & German

www.testingexperience.learntesting.com



Advanced Software Test Design Techniques Decision Tables and Cause-Effect Graphs

by Rex Black

The following is an excerpt from my recently-published book, Advanced Software Testing: Volume 1. This is a book for test analysts and test engineers. It is especially useful for ISTQB Advanced Test Analyst certificate candidates, but contains detailed discussions of test design techniques that any tester can—and should—use. In this first article in a series of excerpts, I start by discussing the related concepts of decision tables and cause-effect graphs.

State-Based Testing and State Transition Diagrams

After our discussion of equivalence partitioning and boundary value analysis, I said we would cover three techniques that would prove useful for testing business logic, often more useful than equivalence partitioning and boundary value analysis. We covered decision tables, which are best in transactional testing situations. We covered use cases, where preconditions and postconditions help to insulate one workflow from the previous workflow and the next workflow.

Now, we move on to state–based testing. State-based testing is ideal when we have sequences of events that occur and conditions that apply to those events, and the proper handling of a particular event/condition situation depends on the events and conditions that have occurred in the past. In some cases, the sequences of events can be potentially infinite, which of course exceeds our testing capabilities, but we want to have a test design technique that allows us to handle arbitrarily-long sequences of events.

The underlying model is a state transition diagram or table. The diagram or table connects beginning states, events, and conditions with resulting states and actions.

In other words, some status quo prevailed and the system was in a current state. Then some event occurs, some event that the system must handle. The handling of that event might be influenced by one or more conditions. The event/condition combination triggers a state transition, either from the current state to a new state or from the current state back to the current state again. In the course of the transition, the system takes one or more actions.

Given this model, we generate tests that traverse the states and transitions. The inputs trigger events and create conditions, while the expected results of the test are the new states and actions taken by the system.

Various coverage criteria apply for state based testing. The weakest criterion requires that the tests visit every state and traverse every transition. We can apply this criterion to state transition diagrams. A higher coverage criterion is at least one test cover every row in a state transition table. Achieving "every-row" coverage will achieve "every state and transition" coverage, which is why I said it was a high coverage criterion.

Another potentially higher coverage criterion requires that at least one test cover each transition sequence of N or less length. The N can be 1, 2, 3, 4, or higher. This is called alternatively "Chow's switch coverage"—after the Professor Chow who developed it—or "N-1 switch coverage," after the level given to the degree of coverage. If you cover all transitions of length one, then "N-1 switch coverage" means "0 switch coverage." Notice that this is the same as the lowest level of coverage discussed. If you cover all transitions of length one and two, then "N-1 switch coverage" means "1 switch coverage". This is a higher level of coverage than the lowest level, of course.

Now, "1 switch coverage" is not necessarily a higher level of coverage than "every-row" coverage. This is because the state transition table forces testing of state and event/condition combinations that do not occur in the state-transition diagram. The so-called "switches" in "N-1 switch coverage" are derived from the state transition diagram, not the state transition table.

You might find all this a bit confusing if you're fuzzy on the test design material covered at the Foundation level. This is a common problem for those who took "brain cram" courses to prepare for the Foundation exam. Don't worry, though, it will be clear to you shortly.

So, what is the bug hypothesis in state-based testing? We're looking for situations where the wrong action or the wrong new state occurs in response to a particular event under a given set of conditions based on the history of event/condition combinations so far.

ISTQB Glossary

State transition testing: A black box test design technique in which test cases are designed to execute valid and invalid state transitions.

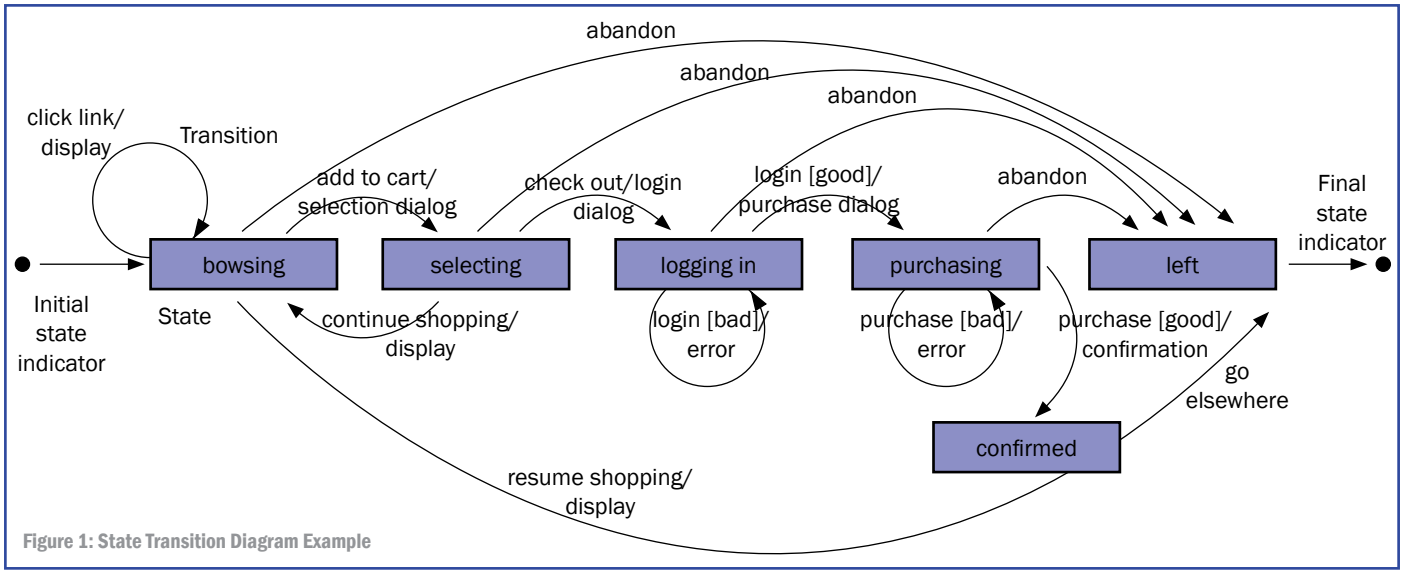


Figure 1: State Transition Diagram Example

Figure 1 shows the state transition diagram for shopping and buying items online from an e-commerce application. It shows the interaction of the system with a customer, from the customer's point of view. Let's walk through it, and I'll point out the key elements of state transition diagrams in general and the features of this one in particular.

First, notice that we have at the left-most side a small dot-and-arrow element labeled "initial state indicator". This notation shows that, from the customer's point of view, the transaction starts when she starts browsing the Web site. We can click on links and browse the catalog of items, remaining in a browsing state. Notice that the looping arrow above the browsing state. The nodes or bubbles represent states, as shown by the label below the browsing state. The arrows represent transitions, as shown by the label above the looping arrow.

Next, we see that we can enter a "selecting" state by adding an item to the shopping cart. "add to cart" is the event, as shown by the label above. The system will display a "selection dialog" where we ask the customer to tell us how many of the item she wants, along with any other information we need to add the item to the cart. Once that's done, the customer can tell the system she wants to continue shopping. In which case, the system displays the home screen again, and the customer is back in a browsing state. From a notation point of view, notice that the actions taken by the system are shown under the event and after the slash symbol, on the transition arrow.

Alternatively, the customer can choose to check out. At this point, she enters a logging-in state. She enters login information. A condition applies to that login information: either it was good or it was bad. If it was bad, the system displays an error and the customer remains in the logging-in state. If it was good, the system displays the first screen in the purchasing dialog. Notice that the "bad" and "good" shown in brackets are, notationally, conditions.

While in the purchasing state, the system will display screens and the customer will enter payment information. Either that information is good or bad—conditions again—which determines whether we can complete and confirm the transaction. Once the transaction is confirmed, the customer can either resume shopping or go somewhere else.

Notice also that the user can always abandon the transaction and go elsewhere.

When we talk about state-based testing during our training courses, people often ask, "How do I distinguish a state, an event, and an action?" The main distinctions are as follows:

> A state persists, until something happens—something external to the thing itself, usually—to trigger a transition. A state can persist for an indefinite period.
>
> An event occurs, either instantly or in a limited, finite period. It is the something that happened—the external occurrence—that triggered the transition.
>
> An action is the response the system has during the transition. An action, like an event, is either instantaneous or requires a limited, finite period.

That said, it is sometimes possible to draw the same situation differently, especially when a single state or action can be split into a sequence of finer-grained states, events, and actions. We'll see an example of that in a moment, splitting the purchase state into substates.

Finally, notice that, at the outset, I said that I drew Figure 1 from the customer's point of view. Notice that, if I drew this from the system's point of view, it would look different. Maintaining a consistent point of view is critical when drawing these charts, otherwise nonsensical things will happen.

State-based testing uses a formal model, so we can have a formal procedure for deriving tests from them. The list below shows a procedure that will work to derive tests that achieve state/transition cover (i.e., "0 switch cover").

1. Adopt a rule for where a test procedure or test step must start and where it may or must end. An example is to say that a test step must start in an initial state and may only end in a final state. The reason for the "may" or "must" wording on the ending part is because, in situations where the initial and final states are the same, you might want to allow sequences of states and transitions that pass through the initial state more than once.
2. From an allowed test starting state, define a sequence of event/condition combinations that leads to an allowed test ending state. For each transition that will occur, capture the expected action that the system should take. This is the expect result.
3. As you visit each state and traverse each transition, mark it as covered. The easiest way to do this is to print the state transition diagram and then use a marker to highlight each node and arrow as you cover it.
4. Repeat steps 2 and 3 until all states have been visited and all transitions traversed. In other words, every node and arrow has been marked with the marker.

This procedure will generate logical test cases. To create concrete test cases, you'd have to generate the actual input values and the actual output values. For this article, I intend to generate logical tests to illustrate the techniques, but remember, as I mentioned before, at some point before execution, the implementation of concrete test case must occur.

Let's apply this process to the example e-commerce application we've just looked at, as shown in Figure 2, where the dashed lines indicate states and transitions that were covered. Here, we see two things:

First, we have the rule that says that a test must start in the initial state and must end in the final state.

Next, we generate the first test step. (browsing, click link, display, add to cart, selection dialog, continue shopping, display, add to cart, selection dialog, checkout, login dialog, login[bad], error, login[good], purchase dialog, purchase[bad], error, purchase[good], confirmation, resume shopping, display, abandon, left).

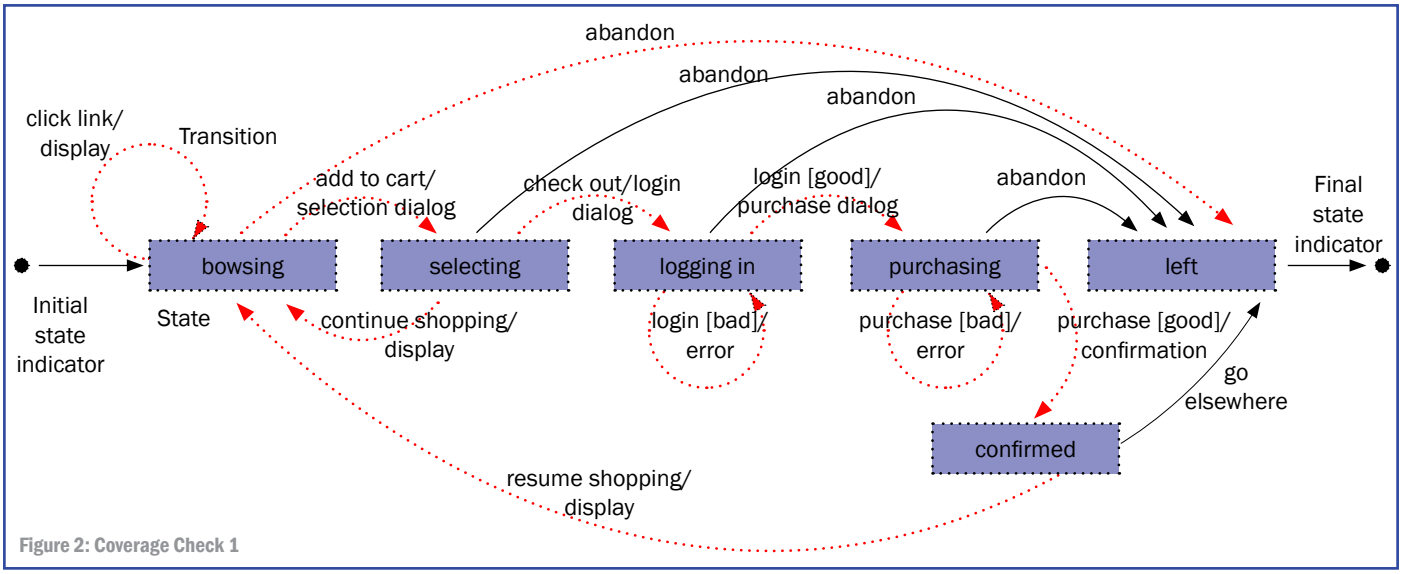


Figure 2: Coverage Check 1

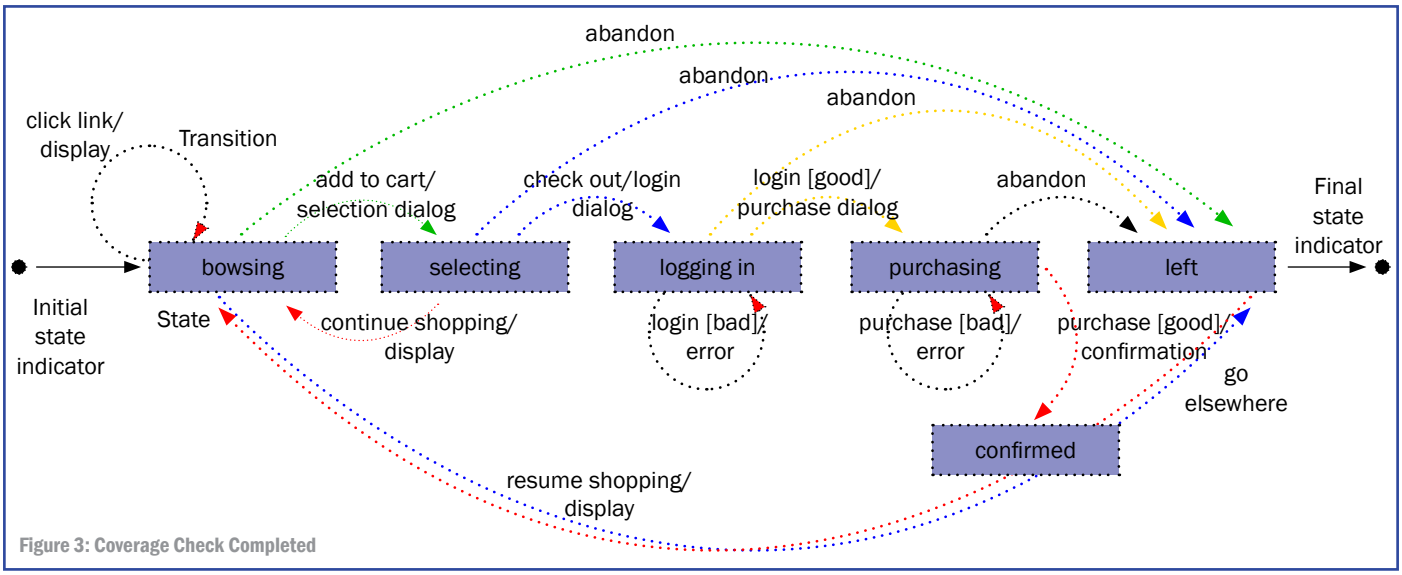


Figure 3: Coverage Check Completed

At this point, we check completeness of coverage, which we've been keeping track of on our state transition diagram.

As you can see, we covered all of the states and most transitions, but not all of the transitions. We need to create some more tests.

Figure 3 shows the test coverage achieved by the following addition test steps, which covers the state transition diagram:

1. (browsing, click link, display, add to cart, selection dialog, continue shopping, display, add to cart, selection dialog, checkout, login dialog, login[bad], error, login[good], purchase dialog, purchase[bad], error, purchase[good], confirmation, resume shopping, display, abandon, left)
2. (browsing, add to cart, selection dialog, abandon, <no action>, left)
3. (browsing, add to cart, selection dialog, checkout, login dialog, abandon, <no action>, left)
4. (browsing, add to cart, selection dialog, checkout, login dialog, login[good], purchase dialog, abandon, <no action>, left)
5. (browsing, add to cart, selection dialog, continue shopping, display, add to cart, selection dialog, checkout, login dialog, login[good], purchase dialog, purchase[good], confirmation, go elsewhere, <no action>, left)

Again, Figure 3 shows the coverage tracing for the states and transitions. You're not done generating tests until every state and every transition has been highlighted, as shown here.

Superstates and Substates

In some cases, it makes sense to unfold a single state into a superstate consisting of two or more substates. In Figure 4, you see that I've taken the purchasing state from the e-commerce example and expanded it into three substates.

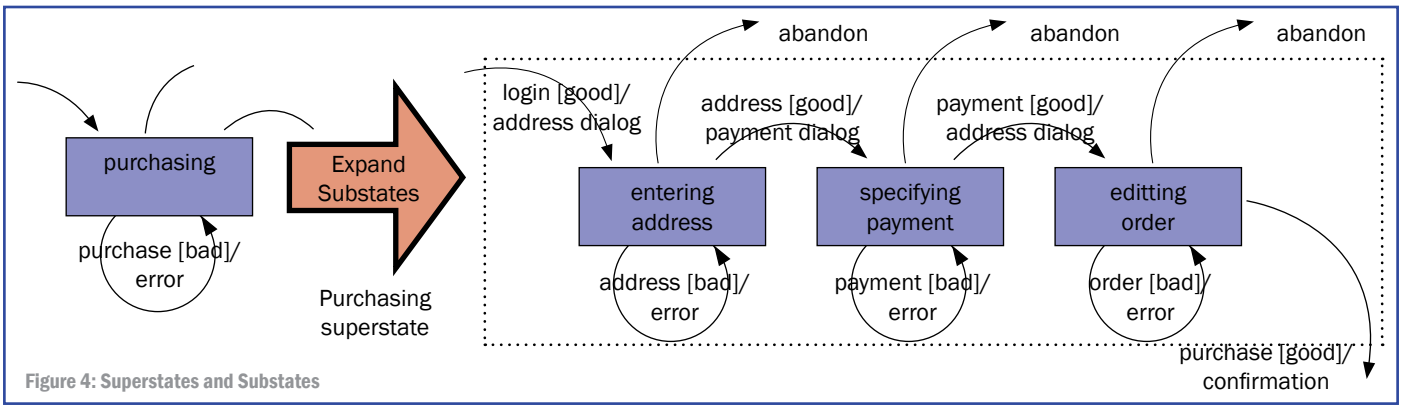


Figure 4: Superstates and Substates

SOFTWARE & SYSTEMS
QUALITY CONFERENCES
United Kingdom | 2009

Interested in the Software Quality Management and Testing world?

- Learn from industry experts speaking about their involvement in real projects
- Hear about the latest trends currently happening in your industry

5 October 2009, London

Go to www.sqs-conferences.com/uk and book your place today

15% for subscribers of Testing Experience

Premium Sponsors:

blackduck

Facilita
software performance testing solutions

hp

sqs

Quality, a vain quest in Crisis Times?

Madrid, 27th-29th of October 2009
South European Conference
on Software Testing
Special Rates for Testing Experience Subscribers

www.expoqa.com/en
Endorsed by:
SOGETI

expoQA

The rule for basic coverage here follows simply. Cover all transitions into the superstate, all transitions out of the superstate, all substates, and all transitions within the superstate.

Notice that, in our example, this would increase the number of tests, because we now have three "abandon" transitions to the "left" state out of the purchasing superstate, rather than just one transition from the purchasing state. This would also add a finer-grained element to our tests—i.e., more events and actions—as well as making sure we tested at least three different types of bad purchasing entries.

State Transition Tables

State transition tables are useful because they force us—and the business analysts and the system designers—to consider combinations of states with event/condition combinations that they might have forgotten.

To construct a state transition table, you first list all the states from the state transition diagram. Next, you list all the event/condition combinations shown on the state transition diagram. Then, you create a table that has a row for each state with every event/condition combination. Each row has four fields:

- Current state
- Event/condition
- Action
- New state

For those rows where the state transition diagram specifies the action and new state for the given combination of current state and event/condition, we can populate those two fields from the state transition diagram. However, for the other rows in the table, we have found undefined situations.

We can now go to the business analysts, system designers, and other such people and ask, "So, what exactly should happen in each of these situations?"

You might hear them say, "Oh, that can never happen!" As a test analyst, you know what that means. Your job now is to figure out how to make it happen.

You might hear them say, "Oh, well, I'd never thought of that." That probably means you just prevented a bug from ever happening, if you are doing test design during system design.

Figure 5 shows an excerpt of the table we

				Current State	Event/cond	Action	New State
				Browsing	Click link	Display	Browsing
		Click link		Browsing	Add to cart	Selection dia	Selecting
		Add to cart		**Browsing**	**Continue shopping**	**Undefined**	**Undefined**
		Continue shopping		**Browsing**	**Check out**	**Undefined**	**Undefined**
Browsing		Check out		**Browsing**	**Login [bad]**	**Undefined**	**Undefined**
Selecting		Login [bad]		**Browsing**	**Login [good]**	**Undefined**	**Undefined**
Logging in		Login [good]		**Browsing**	**Purchase [bad]**	**Undefined**	**Undefined**
Purchasing		Purchase [bad]		**Browsing**	**Purchase [good]**	**Undefined**	**Undefined**
Confirmed	X	Purchase [good]	=	**Browsing**	**Abandon**	**<no action>**	**Left**
Left		Abandon		**Browsing**	**Resume shopping**		**Undefined**
		Resume shopping		**Browsing**	**Go elsewhere**		**Undefined**
		Go elsewhere		**Selecting**	**Click link**		**Undefined**
				{Fifty-three rows, generated in the pattern shown above, not shown}			
				Left	**Go elsewhere**	**Undefined**	**Undefined**

Figure 5: State Transition Table Example

would create for the e-commerce example we've been looking at so far. We have six states:

- Browsing
- Selecting
- Logging in
- Purchasing
- Confirmed
- Left

We have eleven event/condition combinations:

- Click link
- Add to cart
- Continue shopping
- Check out
- Login[bad]
- Login[good]
- Purchase[bad]
- Purchase[good]
- Abandon
- Resume shopping
- Go elsewhere

That means our complete state transition table would have sixty-six rows, one for each possible pairing of a specific state with a specific event/condition combination.

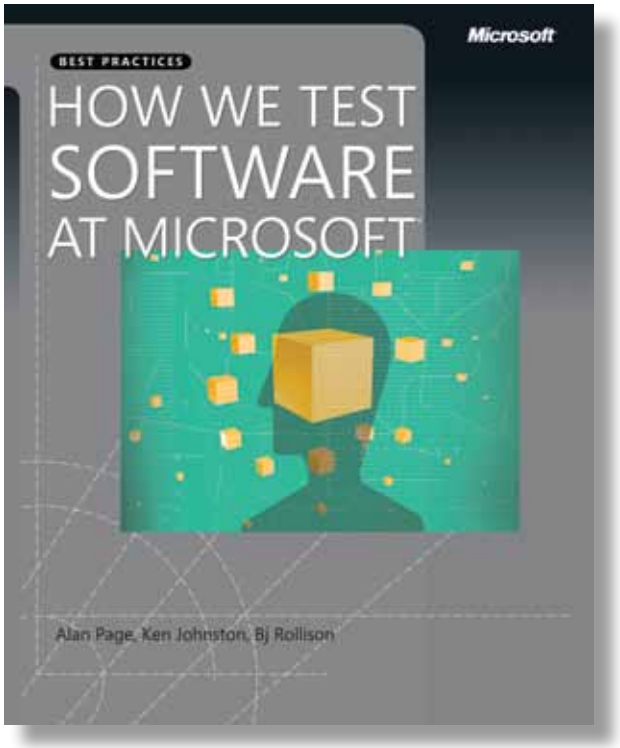
To derive a set of tests that covers the state transition table, we can follow the procedure shown in the steps below. Notice that we build on an existing set of tests created from the state transition diagram to achieve state/transition or 0-switch cover

1. Start with a set of tests (including the starting and stopping state rule), derived from a state transition diagram, that achieves state/transition covered.
2. Construct the state transition table and confirm that the tests cover all the defined rows. If they do not, then either you didn't generate the existing set of tests properly, or you didn't generate the table properly, or the state transition diagram is screwed up. Do not proceed until you have identified and resolved the problem, including re-creating the state transition table or the set of tests, if necessary.
3. Select a test that visits a state for which one or more undefined rows exists in the table. Modify that test to attempt to introduce the undefined event/condition combination for that state. Notice that the action in this case is undefined.
4. As you modify the tests, mark the row as covered. The easiest way to do this is to take a printed version of the table and use a marker to highlight each row as covered.
5. Repeat steps 3 and 4 until all rows have been covered.

Again, this procedure will generate logical test cases.

INTRODUCING

- Discover how Microsoft implements and manages the software-testing process company-wide
- Gain expert insights on effective testing techniques and methodologies
- Shares such facts as the number of test machines at Microsoft, how the company uses automated test cases, and bug statistics
- Answers key testing questions, such as who tests what, when, and with what tools
- Describes how test teams are organized, when and how testing gets automated, testing tools, and feedback



HOW WE TEST SOFTWARE AT MICROSOFT
ISBN: 9780735624252

ABOUT THE AUTHORS

Alan Page has been a software tester for more than 14 years, including more than 12 years at Microsoft Corporation.
Ken Johnston is group manager for testing in the Microsoft Office business group.
Bj Rollison is a Test Architect in the test excellence organization at Microsoft.
Authors' website: http://www.hwtsam.com

Microsoft® Press

Buy the book! **http://www.microsoft.com/learning/en/us/books/11240.aspx**

- Existing test
  - (browsing, add to cart, selecting dialog, checkout, login dialog, login [good], purchase dialog, abandon, <no action>, left)
- Modified test
  - (browsing, *attempt*: continue shopping, *action undefined*, add to cart, selection dialog, checkout, login dialog, login [good], purchase dialog, abandon, <no action>, left)
  - (browsing, *attempt*: check out, *action undefined*, add to cart, selection dialog, checkout, login dialog, login [good], purchase dialog, abandon, <no action>, left)
  - There are six other modified tests for browsing, not shown…
- Don't try to cover undefined event/conditions combinations for more than one state in any test, because you don't know wheter the system will remain testable!
- Best case scenario is that the undefined event/condition combination is ignored or rejected with an intelligent error message, and processing continues normally from there

Figure 6: Deriving Tests Example

Figure 6 shows an example of deriving table-based tests, building on the e-commerce example already shown. At the top, you can see that I've selected an existing test from the larger set of tests derived before.

> (browsing, add to cart, selection dialog, checkout, login dialog, login[good], purchase dialog, abandon, <no action>, left)

Now, from here I started to create modified tests to cover undefined browsing event/conditions, and those undefined conditions only.

One test is: (browsing, *attempt*: continue shopping, *action undefined*, add to cart, selection dialog, checkout, login dialog, login[good], purchase dialog, abandon, <no action>, left)

Another test is: (browsing, *attempt*: check out, *action undefined*, add to cart, selection dialog, checkout, login dialog, login[good], purchase dialog, abandon, <no action>, left)

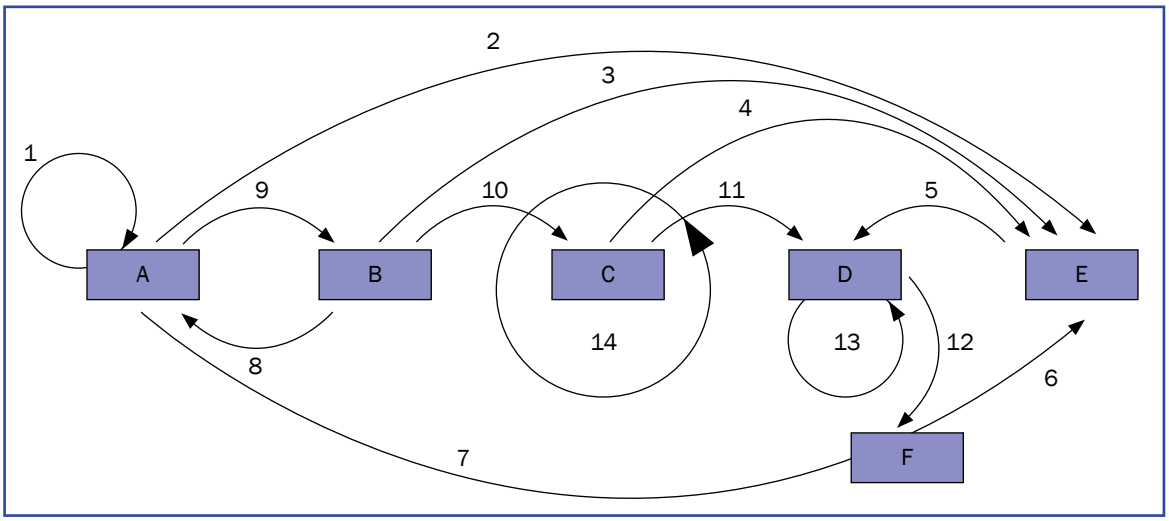
There are six other modified tests for browsing, which I've not shown. As you can see, it's a mechanical process to generate these tests. As long as you are careful to keep track of which rows you've covered—using the marker trick I mentioned earlier, for example—it's almost impossible to forget a test.

Now, you'll notice that I only included one undefined event/condition combination in each test step. Why? This is a variant of the equivalence partitioning rule that we should not create invalid test cases that combine multiple invalids. In this case, each row corresponds to an invalid. If we try to cover two rows in a single test step, we can't be sure the system will remain testable after the first invalid.

Notice that I indicated that the action is undefined. What is the ideal system behavior under these conditions? Well, the best-case scenario is that the undefined event/condition combination is ignored or—better yet—rejected with an intelligent error message. At that point, processing continues normally from there. In the absence of any meaningful input from business analysts, the requirements specification, system designers, or any other authority, I would take the position that any other outcome is a bug, including some inscrutable error message like, "What just happened can't happen." (No, I'm not making that up. An RBCS course attendee once told me she had seen exactly that message when inputting an unexpected value.)

Switch Coverage

Figure 7 shows how we can generate sequences of transitions, using the concept of switch coverage. I'm going to illustrate this concept with the e-commerce example we've used so far.



0-switch			1-switch								
A1	A2	A9	A1A1	A1A2	A1A9				A9B10	A9B8	A9B3
B10	B8	B3	B10C14	B10C11	B10C4	B8A1	B8A2	B8A9			
C14	C11	C4	C14C14	C14C11	C14C4	C11D13	C11D12	C11D5			
D13	D12	D5	D13D13	D13D12	D13D5	D12F6	D12F7				
F6	F7					F7A1	F7A2	F7A9			

Figure 7: N-1 Switch Coverage Example

At the top of Figure 7, you see the same state transition as before. Except, I have replaced the state labels with letters, and the transition labels with numbers. Now, a state/transition pair can be specified as a letter followed by a number. Notice that I'm not bothering to list, in the table below, a letter after the number, because it's unambiguous from the diagram what state we'll be in after the given transition. There is only one arrow labeled with a given number that leads out of a state labeled with a given letter, and that arrow lands on exactly one state.

The table contains two types of columns. The first is the state/transition pairs that we must cover to achieve 0-switch coverage. Study this for a moment, and assure yourself that, by designing tests that cover each state/transition pair in the 0-switch columns, you'll achieve state/transition coverage as discussed previously.

Constructing the 0-switch columns is easy. The first row consists of the first state, with a column for each transition leaving that state. There are at most three transitions from the A state. Repeat that process for each state for which there is an outbound transition. Notice that the E state doesn't have a row, because E is a final state and there's no outbound transition. Notice also that, for this example, there are at most three transitions from any given state.

The 1-switch columns are a little trickier to construct, but there's a regularity here that makes it mechanical if you are meticulous. Notice, again, that after each transition occurs in the 0-switch situation, we are left in a state, which is implicit in the 0-switch cells. As mentioned above, there are at most three transitions from any given state. So, that means that, for this example, each 0-switch cell can expand to at most three 1-switch cells.

So, we can take each 0-switch cell for the A row and copy it into three cells in the 1-switch columns, for nine cells for the A row. Now, we ask ourselves, for each triple of cells in the A row of the 1-switch columns, what implicit state did we end up in? We can then refer to the appropriate 0-switch cells to populate the remainder of the 1-switch cell.

Notice that the blank cells in the 1-switch columns indicate situations where we entered a state in the first transition from which there was no outbound transition. In this example, that is the state labeled "E" in Figure 7, which was labeled "Left" on the full-sized diagram.

So, given a set of state/transition sequences like those shown—whether 0-switch, 1-switch, 2-switch, or even higher—how do we derive test cases to cover those sequences and achieve the desired level of coverage? Again, I'm going to build on an existing set of tests created from the state transition diagram to achieve state/transition or 0-switch cover.

1. Start with a set of tests (including the starting and stopping state rule), derived from a state transition diagram, that achieves state/transition coverage
2. Construct the switch table using the technique shown previously. Once you have, confirm that the tests cover all of the cells in the 0-switch columns. If they do not, then either you didn't generate the existing set of tests properly, or you didn't generate the switch table properly, or the state transition diagram is screwed up. Do not proceed until you have identified and resolved the problem, including re-creating the switch table or the set of tests, if necessary. Once you have that done, check for higher-order switches already covered by the tests.
3. Now, using 0-switch sequences as needed, construct a test that reaches a state from which an uncovered higher-order switch sequence originates. Include that switch sequence in the test. Check to see what state this left you in. Ideally, another uncovered higher-order switch sequence originates from this state, but, if not, see if you can use 0-switch sequences to reach such a state. You're crawling around in the state transition diagram looking for ways to cover higher-order sequence. Repeat this for the current test until the test must terminate.
4. As you construct tests, mark the switch sequences as covered once you include them in a test. The easiest way to do this is to take a printed version of the switch table and use a marker to highlight each cell as covered.
5. Repeat steps 3 and 4 until all switch sequences have been covered.

Again, this procedure will generate logical test cases.

In Figure 7, we see the application of the derivation technique covered previously to the e-commerce example we've used. After finishing the second step, that of assessing coverage already attained via 0-switch coverage, we can see that most of the table is already populated. Those are the blue-shaded cells shown in Figure 8, which are covered by the five existing state/transition tests.

Now, we generate five new tests to achieve 1-switch coverage. Those are listed below and shown in Figure 8 as red-shaded cells.

- (A1A1A2).
- (A9B8A1A9B8A2).
- (A9B10C14C14C4).
- (A9B10C11D13D13D5).
- (A9B10C11D12F7A1A9B10C11D12F7A9).

Let me mention something about this algorithm for deriving higher-order switch coverage tests, as well as the one given previously for row-coverage tests. Both build on an existing set of tests that achieve state/transition coverage. That is efficient from a test design point of view. It's also conservative from a test execution point of view, because we cover the less challenging stuff first and then move on to the more difficult tests.

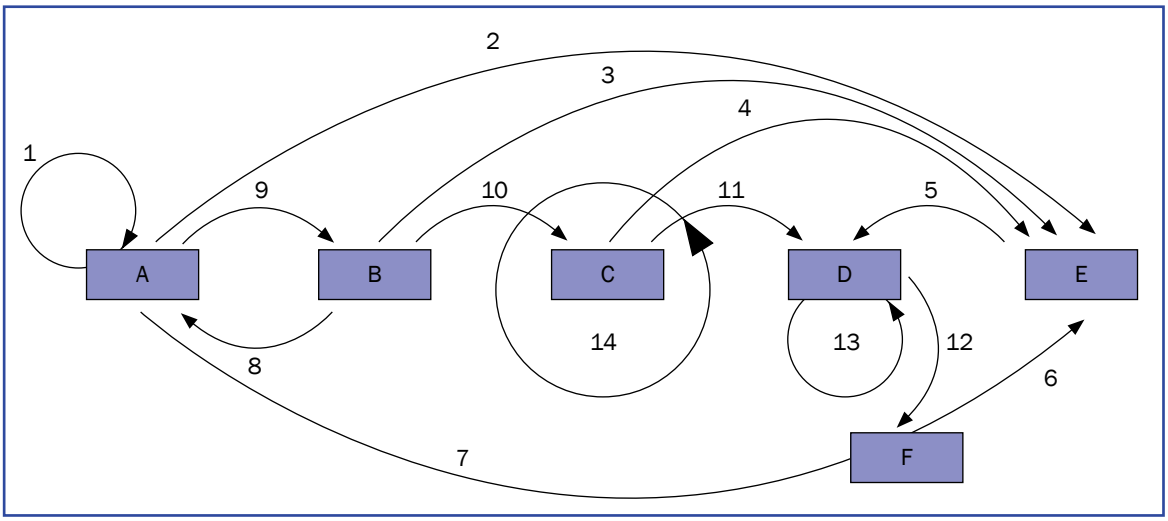


Figure 7: N-1 Switch Coverage Example [1]

0-switch			1-switch								
A1	A2	A9	A1A1	A1A2	A1A9				A9B10	A9B8	A9B3
B10	B8	B3	B10C14	B10C11	B10C4	B8A1	B8A2	B8A9			
C14	C11	C4	C14C14	C14C11	C14C4	C11D13	C11D12	C11D5			
D13	D12	D5	D13D13	D13D12	D13D5	D12F6	D12F7				
F6	F7					F7A1	F7A2	F7A9			

Figure 7: N-1 Switch Coverage Example [2]

However, it is quite possible that, starting from scratch, a smaller set of tests could be built, both for the row coverage situation and for the 1-switch coverage situation. If the most important thing is to create the minimum number of tests, then you should look for ways to reduce the tests created, or modify the derivation procedures given here to start from scratch rather than to build on an existing set of 0–switch tests.

State Testing with Other Techniques

Let's finish our discussion on state-based testing by looking at a couple interesting questions. First, how might equivalence partitioning and boundary value analysis combine with state-based testing? The answer is, quite well.

From the e-commerce example, suppose that the minimum purchase is $10 and the maximum is $10,000. In that case, we can perform boundary value analysis as shown in Figure 9, performed on the purchase[good] and purchase[bad] event/condition combinations. By covering not only transitions, or rows, or transitions sequence, but also boundary values, this forces us to try different purchase amounts.

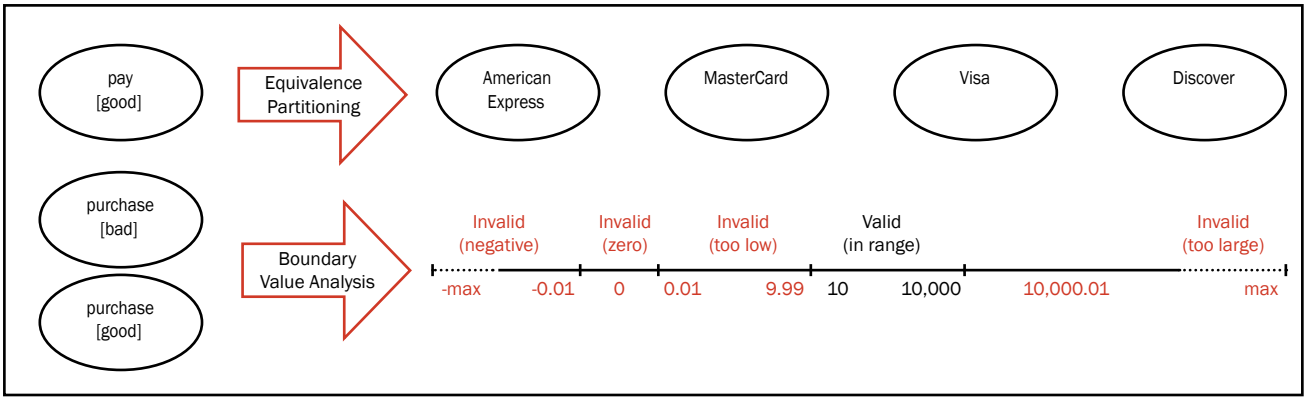


Figure 9: Equivalence Partitions and Boundary Values

We can also apply equivalence partitioning to the pay[good] event/condition combination. For example, suppose we accept four different types of credit cards. By covering not only transitions, or rows, or transition sequences, but also equivalence partitions, this forces us to try different payment types.

Now, to come full circle on a question I brought up at the start of the discussion on these three business-logic test techniques. When do we use decision tables and when do we use state diagrams?

This can be, in some cases, a matter of taste. The decision table is easy to use and compact. If we're not too worried about the higher-order coverage, or the effect of states on the tests, we can model many state-influenced situations as decision tables, using conditions to model states. However, if the decision table's conditions section starts to become very long, you're probably stretching the technique. Also, keep in mind that test coverage is usually more thorough using state-based techniques. In most cases, one technique or the other will clearly fit better. If you are at a loss, try both and see which feels most appropriate.

Conclusion

In this article, I've shown how to apply state transition diagram, state transition tables, and switch coverage analysis to the testing of sophisticated and complex system, especially embedded systems and other systems where the system response to a given set of inputs depends on what the system has dealt with in the past. We have already looked at decision tables as a way to test detailed business rules, and now state-based methods to test state-dependent systems. However, we will need to look at an additional technique, use cases, to deal with the full range of internal business logic testing we need to do. I'll address that technique in the next articles in this series.

Biography

See page 40.

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
print version: 8,00 €

ISSN 1866-5705

In all publications Díaz & Hilterscheid Unternehmensberatung GmbH makes every effort to respect the copyright of graphics and texts used, to make use of its own graphics and texts and to utilise public domain graphics and texts.

All brands and trademarks mentioned, where applicable, registered by third-parties are subject without restriction to the provisions of ruling labelling legislation and the rights of ownership of the registered owners. The mere mention of a trademark in no way allows the conclusion to be drawn that it is not protected by the rights of third parties.

The copyright for published material created by Díaz & Hilterscheid Unternehmensberatung GmbH remains the author's property. The duplication or use of such graphics or texts in other electronic or printed media is not permitted without the express consent of Díaz & Hilterscheid Unternehmensberatung GmbH.

The opinions expressed within the articles and contents herein do not necessarily express those of the publisher. Only the authors are responsible for the content of their articles.

No material in this publication may be reproduced in any form without permission. Reprints of individual articles available.

Index Of Advertisers

your great escape

GranCanaria.com

Training with a View

Díaz Hilterscheid

also onsite training worldwide in German, English, Spanish, French at
http://training.diazhilterscheid.com/ training@diazhilterscheid.com

"A casual lecture style by Mr. Lieblang, and dry, incisive comments in-between. My attention was correspondingly high. With this preparation the exam was easy."
Mirko Gossler, T-Systems Multimedia Solutions GmbH

"Thanks for the entertaining introduction to a complex topic and the thorough preparation for the certification. Who would have thought that ravens and cockroaches can be so important in software testing"
Gerlinde Suling, Siemens AG

- subject to modifications -

16.09.09-18.09.09	Certified Professional for Requirements Engineering - Foundation Level	German	Berlin
28.09.09-30.09.09	ISSECO® - Certified Professional for Secure Software Engineering	English	Frankfurt/Neu Isenburg
28.09.09-30.09.09	Certified Tester Foundation Level	English	Berlin
05.10.09-09.10.09	Certified Tester Advanced Level - TESTMANAGER	German	Berlin
12.10.09-14.10.09	Certified Tester Foundation Level - Kompaktkurs	German	Berlin
26.10.09-30.10.09	Certified Tester Advanced Level - Testmanager	German	München
02.11.09-06.11.09	Advanced Level Testing - TECHNICAL TEST ANALYST	German	Berlin
09.11.09-12.11.09	Certified Tester Foundation Level	German	Berlin
16.11.09-18.11.09	Certified Tester Foundation Level - Kompaktkurs	German	Berlin
18.11.09-20.11.09	Certified Professional for Requirements Engineering - Foundation Level	German	Berlin
23.11.09-27.11.09	Certified Tester Advanced Level - TEST ANALYST	German	Berlin
30.11.09-04.12.09	Certified Tester Advanced Level - TESTMANAGER	German	Berlin
02.12.09-04.12.09	ISSECO® - Certified Professional for Secure Software Engineering	English	Frankfurt/Neu Isenburg
07.12.09-10.12.09	Certified Tester Foundation Level	German	Frankfurt
14.12.09-16.12.09	Certified Tester Foundation Level - Kompaktkurs	German	Berlin
14.12.09-17.12.09	Certified Tester Foundation Level	German	Düsseldorf/Köln

Kurfürstendamm, Berlin © Katrin Schülke