

PROFESSIONAL TESTER

SUBSCRIBE
It's FREE
for testers

Essential for software testers

| October 2012 | £ 4 / € 5 | v2.0 | number 17 |

t e s t
d e v
o p s

Including articles by:

Stephen Janaway

Ole Lensmar
SmartBear Software

Wolfgang Platz
Tricentis

**Jeanne Hofmans and
Erwin Pasmans**
Improve Quality Services

Martin Mudge
Bugfinders.com

Test Studio

Easily record automated tests for your modern HTML5 apps



Test the reliability of your rich, interactive JavaScript apps with just a few clicks. Benefit from built-in translators for the new HTML5 controls, cross-browser support, JavaScript event handling, and codeless test automation of multimedia elements.

www.telerik.com/html5-testing



 **telerik**



Contact

Editor

Edward Bishop
editor@professionaltester.com

Managing Director

Niels Valkering
ops@professionaltester.com

Art Director

Christiaan van Heest
art@professionaltester.com

Sales

Rikkert van Erp
advertise@professionaltester.com

Publisher

Jerome H. Mol
publisher@professionaltester.com

Subscriptions

subscribe@professionaltester.com



Contributors to this issue

Jeanne Hofmans
Stephen Janaway
Ole Lensmar
Martin Mudge
Erwin Pasmans
Wolfgang Platz

Professional Tester is published
by Professional Tester Inc

We aim to promote editorial independence and free debate: views expressed by contributors are not necessarily those of the editor nor of the proprietors.
©Professional Tester Inc 2012.
All rights reserved. No part of this publication may be reproduced in any form without prior written permission.
“Professional Tester” is a trademark of Professional Tester Inc.

Insert strained cloud pun here

Some testers are sometimes asked for their opinion on operations. Some of them may be wondering whether they should recommend porting applications to cloud. They should.

There are many things that foil testing. They are the things that make our test results unindicative of what will happen in production. That defeats the object of testing and limits its growth. One of them is the test environment. It tends to be different from the development environment because building and maintaining two environments tends to be twice as expensive as building one.

But not with cloud. Infrastructure as a service makes doing exactly that much cheaper. So for cloud applications there is no excuse for a test environment to be technically different in any way from the

development environment. Testers are freed to concentrate on the even more difficult chronological difference: the difference between current test and future production data, including that provided by external systems and/or originating from real-time sensors. This problem is theoretically impossible to solve. But, with the technical problem solved by cloud, we can try. We will never know the future but we can learn more about what bits of it matter most to the indicativeness of our test results and so improve our guesses.

Edward Bishop

Editor



IN THIS ISSUE

TestDevOps

4 Move closer

Lessons learned from a year of change to TestDevOps with Stephen Janaway

8 Closing the TestDevOps gap

Ole Lensmar explains web API testing

12 TestOps

Edward Bishop discusses testing's role in DevOps

16 BizTestDevOps

Wolfgang Platz on practical governance of DevOps via test automation

Looking Inside Testing Services

20 Quality Level Management: who, what and how

Jeanne Hofmans and Erwin Pasmans in interview

Factory model testing

24 Put the crystal ball away

Martin Mudge argues for the use of a pay-per-defect testing service

 Visit professionaltester.com for the latest news and commentary

Move closer

by Stephen Janaway

Don't stop



Stephen Janaway reports on his department's transition to TestDevOps

Independence has advantages. But anything involving the phrase 'someone else's problem' is rarely a good way of working. My superiors decided my department must change the relationship between testing, development and operations, to make it closer and more productive. I agreed. Here is what happened.

The pursuit of happiness

Before that decision, we used a very traditional waterfall methodology. Requirements, usually defined by strangers, were received. Architects designed. Developers coded. Testers tested. While (no one stopped them) {Developers fixed defects. Testers retested and regression tested}. Then the products were packaged for delivery downstream. Step by step, tick, tick, tick, handover by handover.

This required sysadmins, build managers and release managers. Their task was not

easy: after the first release, a big bang integration of two weeks of development work happened every two weeks, never without problems. The operators, quietly and efficiently, kept the build farms and tools running and updated. To the developers and testers, operators and their functions were transparent and rarely considered except on the rare occasions the operators had no choice but to request maintenance downtime of a dev, test, or the production, environment. Indeed, these occasions were the reason for their only contact with developers or testers.

But the ops were unhappy. They were spending an increasing proportion of the two weeks on the integration, pressurizing their own work, the vital necessity for which is caused by change not to the application under test but to third-party infrastructure (including third-party software), time and throughput.

The developers were unhappy too. They were losing coding time investigating incidents which turned out to be simple integration issues.

Unsurprisingly, the testers were least happy. They could not execute their newly maintained tests even in the production environment, let alone the test environments which were given much lower priority, until the next new integration, which would require further test maintenance, was nearly due.

This unhappiness continued until eventually the directors became unhappy too. Their decision, broadcast to all concerned, was: we are going agile.

Testers winced. They were used to working with too little and too poor input (documentation). Now they worried about getting even less and worse.

Operators winced. They were used to working with too poor input (software). Now they worried about getting even worse, and more frequently.

To make the transition happen and successful, a cultural transition was needed: the realization that agile is not about doing away with documentation, nor with changing production more frequently, but working more closely together. To achieve that we (testers, developers and operators) should not *do* one another's work, but strive better to *understand* it.

This of course is easier said than done. We struggled with it, especially our build managers who were hit by a tsunami of completed but unintegratable "tasks" stressing their build farms and accompanied by cries of "it built fine for me" or "the problem is caused by another task which is wrong, speak to its author".

Then we found our holy grail, the thing that could make everyone happy.

Continuous integration using Jenkins

The developers liked it because they could see their enhancements working sooner. The build managers liked it because it allowed them to get the incidents they raised resolved faster. The sysadmins liked it because it provided them with a shopping list of new infrastructure and tools they needed to implement it. The directors liked it because it gave them a management dashboard with meters, coloured lights and simple controls.

The testers liked it because it gave them builds worth testing sooner. But slowly, by studying the work of the build managers and sysadmins, the testers learned how to use it themselves. They started to *control* the builds, as testing should. As the build approached adequate quality, testing acted as a formal gateway, with all entailing responsibilities, to downstream delivery.

The secret of our success

A year later, we're convinced that the change has made the department more efficient and most if not all individuals

happier. There is a lot more informal communication: roles that did not talk to one another before now do, even sysadmins and directors occasionally. Whether all this talk is a good thing remains an open question, but I think so because it is giving team members more appreciation of what their teammates do. This tends to help everyone to make a contribution to every release.

If you believe as I do that testers should foster close relationships with developers, and developers believe they should do so with operators, it seems logical that testers should do so also. Thus we understand *why* builds are late, items pass testing then fail in production and so on, and this leads to opportunities for us to help, take more responsibility, improve process and so on, keeping testing where it should always be: at the centre of things. To achieve that in a TestDevOps environment, testers must develop the traits and skills of the other

groups. For example, a sysadmin needs to be likeable, present ideas well and "get things done". A tester, when working closely with sysadmins, needs to be able to behave like them. I think our experience endorses the common theory that interpersonal relationships, built upon understanding of others and their roles, are more important than tools or techniques.

We made some mistakes

Being encouraged to talk more caused some testers to complain and criticize more. That didn't make late builds arrive any faster or be any better: finding ways to help with them did. For example, before we introduced CI we smoke tested pre-builds, enabling the build managers to integrate changes in stages and to concentrate on the next stage rather than testing the previous one. It also helped us to understand how to use CI, which we now do fluently, to great advantage to our testing. If you are a tester in an organization



moving towards TestDevOps, start by learning more about the process by which products are delivered to you.

We assumed that all testers could and wanted to get involved at a very technical level. Beware of allowing good testers who find that difficult or uninteresting to become worried about it. Testing is a complex discipline with its own specializations. It's not necessary nor, arguably, desirable for a person capable of performing one of them to spend effort becoming also an expert developer and operator. There is a part of testing which is pure testing and should not be subject to influence from other roles.

Once we had CI working well for us, we rested on our laurels for a little while and that was wrong. Having helped to achieve greater automation of dev and ops, so that they could integrate better, we should have realized we now needed greater auto-

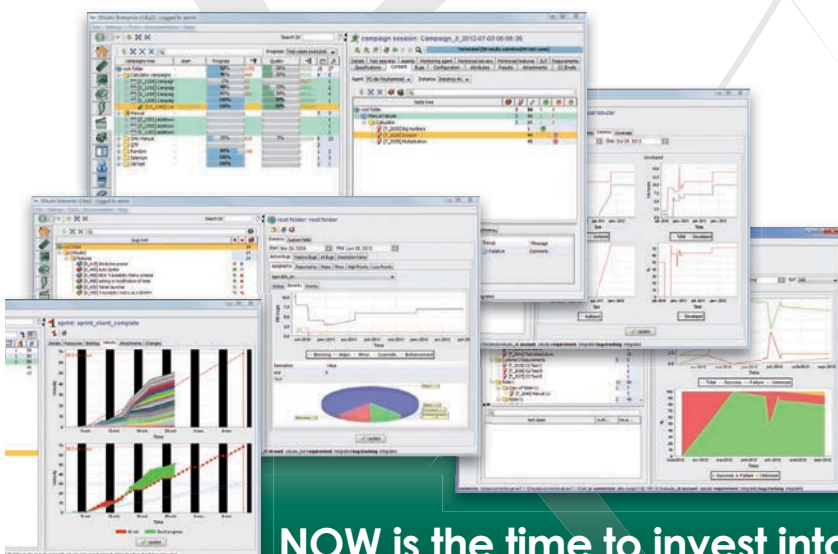
mation of test, for the same reason. Not having it was dangerous because needed builds still sometimes failed and needed test environments were still sometimes unavailable: testing was not ready for the improved process to fail. When we finally did get all the things we needed we set about manual testing that should have been automated. When the things changed, which was now frequent, we had to start that manual testing again. It got a bit like *Groundhog Day*.

We were saved by an improvement in operations, automated deployment, which

lets testers pick up a device with the correct build ready to test at any time. But in the period when we had CI but not AD, significant testing time was lost. It's vital to remember that process improvement, once started, is continuous: if you are not ready to continue it, it's better not to start it. TestDevOps, which I see as a process improvement initiative, is largely about automation because automation is the common ground between the three roles. Testing in TestDevOps must never stop seeking to increase and improve test automation, so that more of testing can be integrated with dev and ops ■

Stephen Janaway is a test manager who has worked for Ericsson, Motorola and Nokia. He blogs on testing at <http://stephenjanaway.co.uk>. Jenkins is available free at <http://jenkins-ci.org>

The crisis will **MAKE** you **WISER!**



- > Web Access
- > Requirements, Tests, Bugs
- > Schedule / Continuous integration
- > Versioning
- > Customizable
- > More than 40 automation test frameworks supported
- > Manual Tests

NOW is the time to invest into a REALLY affordable tool.

XStudio, the revolutionary ALM/Test Management solution
for only \$180 per user.

www.xqual.com



IT Training and
Professional Development



Agile is going places – are you?

BCS, The Chartered Institute for IT, offers the leading agile testing certification for software testers – Certified Agile Tester.

bcs.org/agiletester



01793/PDS/AD/0812

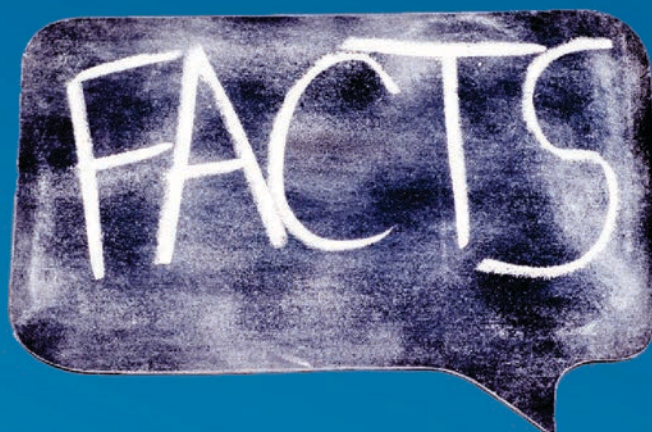
The Certified Agile Tester scheme is a trademark of iSQL.

© BCS, The Chartered Institute for IT, is the business name of The British Computer Society (Registered charity no. 292786) 2012

odin

AXE 3.4 RELEASED!

AXE - A TEST AUTOMATION SOLUTION THAT ACTUALLY DELIVERS



15% Success
11% Tool Incompatibility
17% Insufficient Budget
20% Lack of Experience
37% Lack of Time

Source Data: IDT

85% OF ALL TRADITIONAL SOFTWARE TEST
AUTOMATION INITIATIVES FAIL!

The Axe platform introduces a new class of test automation technology which simplifies test automation tasks, delivering fast results, easy maintenance and quicker return on investment. A complete code and documentation generation platform, aimed at testers and leveraging existing tool investment, that will revolutionise your test automation regime.

FREE WHITE PAPER DOWNLOAD:

www.odintech.com/downloads



Closing the TestDevOps gap

by Ole Lensmar

From test reuse to test unification



Ole Lensmar explains the special aims and requirements of testing web APIs

Most software is unfinished. As long as it remains in service it is subject to change, and probably will change, in response to new requirements and newly discovered defects. Changes made before deployment are considered good because they reduce the need to make changes after deployment where they are considered bad because they can cause regression. The tension between good and bad change is especially high in the rapidly growing domain of web APIs, which inhabit a much more dynamic environment than most software, including non-web APIs. They are more sensitive to environmental change because they usually have many external dependencies, and their rate of change is higher due to market change, technology change and other forces.

That's why functional and performance monitoring, tuned to the end user experience, is now part of pre-deployment development testing, and why pre-deployment development testing is closely tied to post-

deployment functional and performance monitoring. The concept of pre-versus-post deployment testing has already become meaningless. The quality attributes tested before deployment are chosen for their importance after it, and are the same as those tested when subsequent change occurs. The difference is between the testing practices and tools used at different points in the lifecycle. I think of this as testing's own "DevOps gap". In this article I will discuss how to close it.

The key is to use exactly the same test assets both pre- and post-deployment. Otherwise, there is no way to assess the realism of the lab tests and, if any of them fail in the production environment, no way to tell which. The test environment and tools must make it possible to:

- reuse pre-deployment test assets for post-deployment monitoring
- monitor key functional transactions and performance metrics continuously during pre-deployment testing
- add assertions to tests, for example defining rules about the validity of an element in an XML-encoded response to a given input
- add realism to tests, for example by data-driven testing, inserting varying "think time" and simulating attempted SQL injection.

Assert yourself

Luckily, the nature of web APIs helps with the task of designing a gap-free methodology. By definition, any API-accessible web application comes with its own "command line", ie the API itself. That removes one reason why pre-deployment software test assets might need to be

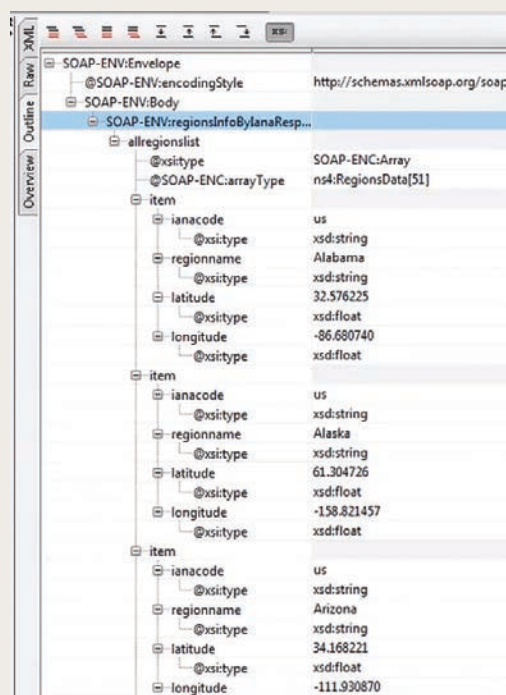


Figure 1: Response message

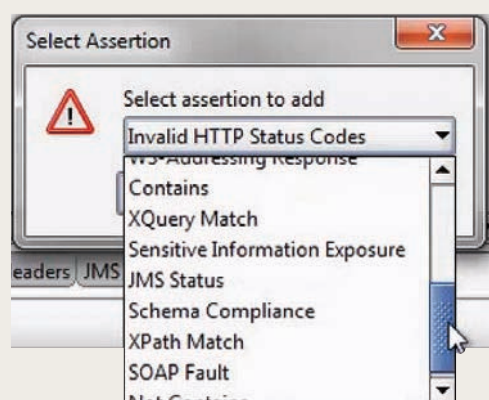


Figure 2: Assertion types

different from post-deployment software test assets. In many non-web situations, developers create a different version of the software specifically to add command lines and create custom scripts whose only purpose is to test against those command lines. After deployment, different test assets are needed to monitor the “real” software from an end-user’s perspective, for example additional custom scripts to run against a user interface.

Moving from pre-deployment to post-deployment testing the API doesn’t need to change, so there is also no need to change the test script. Every web API exposes operations and/or resources,

each of which responds to a particular request: SOAP, REST, HTTP, AMF etc. The response should not change when the API is deployed so neither should test steps.

In functional testing, the correctness of the response is checked using assertions. Here is how that is done using the Open Source tool soapUI. SoapUI generates the requests automatically from the WSDL file. Clicking a request displays it and the API’s response message to it (see figure 1). To create an assertion, right-click any element in the response and select the assertion type (figure 2), then use the configuration panel which appears (figure 3) to define your assertion. For

example, an “XPath Match” would be used to assert that a target element contains a particular value: otherwise, the test will fail.

Be realistic

Assertions don’t change at deployment because the logic exposed by the APIs does not change. What does change is the environment. That includes the underlying data layer, so it is important to craft test assets so that they are not affected by eventual differences between pre- and post-deployment data environments.

The tester’s role of course is to design tests that are realistic: in other words, that predict production conditions with sufficient accuracy to detect all important defects before deployment. For web APIs, completely vulnerable to the unknown behaviour of the systems that use them, that’s especially difficult to achieve and so it’s common, even typical, for web APIs to pass testing but fail in production. But the more realistic the tests, the fewer the unexpected failures and therefore the better their cost and risk (both project and product) are contained. Experience shows that the most likely causes of failure despite effective functional testing are related to unexpected volume, load and security. Therefore realistic testing must include these test types.

Volume testing is best achieved by the data-driven approach. Whereas in functional testing that is used to improve coverage and exhaustiveness, here the aim is to discover requests that cause the APIs to read, process and/or update huge numbers of records from an external data source, and measure how well APIs cope with them.

Testing under load aims to measure how well APIs perform when the numbers of simultaneous requests, and of those with various characteristics, fluctuate: especially when they rise. The most stringent tests are often engineered to include the high demand requests discovered during volume testing. LoadUI

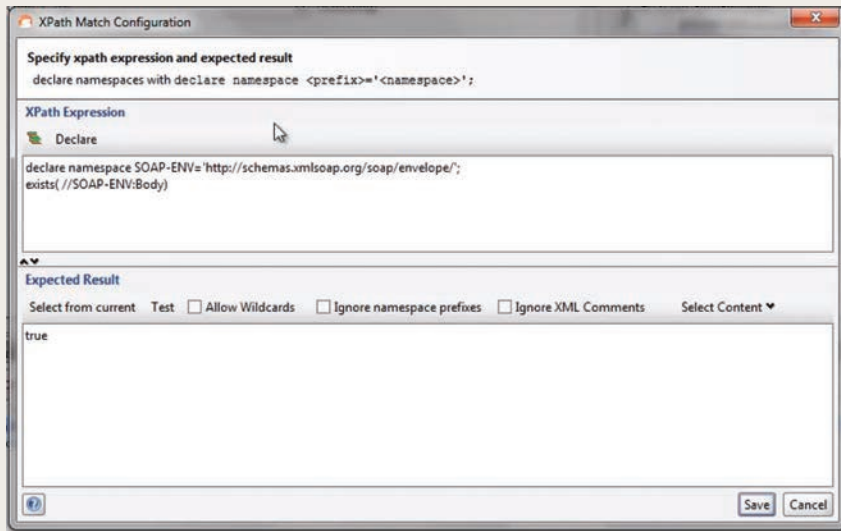


Figure 3: Assertion configuration

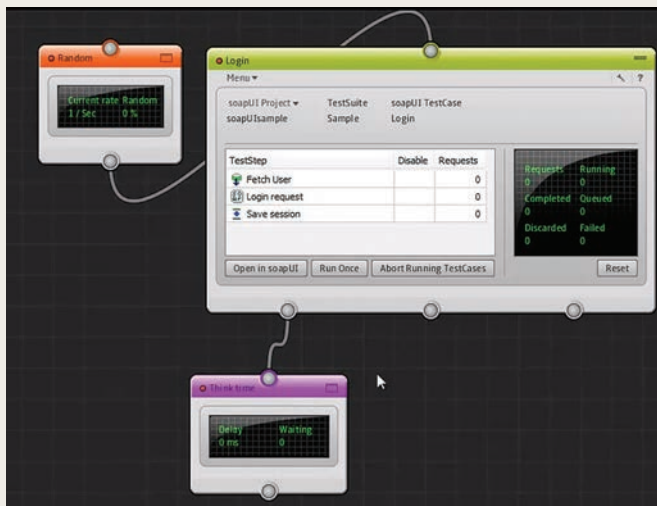


Figure 4: Meter and controller added to a functional test in loadUI

reuses soapUI's functional tests for this purpose and allows meters (for better result capture) and control modules (to improve realism) to be connected to them (figure 4).

As well as executing tests derived from predicted deliberate attacks, for example well-known SQL injection strings such as ' or '1'=1, web API security testing should aim to detect the presence of inappropriate data in responses to innocent, but failure-causing, requests: common examples include stack traces and third-party software version/configuration information. Both are easier when testing web APIs than other software types, because all inputs are encapsulated in requests rather than more complicated things such as user navigation behaviour, and because negative as well as positive

assertions can be used. For example, all tests can include the assertion that the string "Microsoft Windows" does *not* appear in the response. This might lead to some false positives, but they will usually simply confirm that the assertion is working as intended to protect against actual failure.

Turn on, tune in and don't drop out

Unifying pre- and post-deployment testing brings benefit that works in the opposite direction. The results of post-deployment

testing are fed back to pre-deployment test design, especially usefully when requirements and/or design change. Understanding the behaviour of APIs in the actual transactional context can also drive operational decisions such as how to re-cluster infrastructure or provision cloud resources. Using the same tests created during development for post-production monitoring gives testers and developers a head start on using and reacting to monitoring because, having designed the tests and resolved past incidents raised from them, they know the context and meaning of its results.

To achieve this, the tests are deployed to the production environment, that is the web. For example, soapUI tests can be executed on more than 80 monitoring sites worldwide comprising SmartBear's AlertSite network, providing full step- and assertion-level results plus response times and other statistics, from actual production conditions. Applying load in production for testing purposes can cause performance or even reliability failure, so careful load design and scheduling are necessary.

Whether or not tests fail, the comparison of results of the same tests executed pre-deployment, post-deployment and after operational change offers insight into what causes variation, for example infrastructure and network issues, usage patterns, user behaviour or unrealistic test data. Ultimately of course what matters is the service delivered to whatever consumes the service provided by the API and how it affects users. A unified test methodology creates a feedback loop whose resonant frequency is user experience. Everything else, including application, test and monitoring design is tuned to that. It never stops ■

Ole Lensmar is CTO and co-founder of SmartBear Software in Sweden, formerly known as Eviware Software, which created soapUI and was acquired by SmartBear in 2011.

The free, Open Source soapUI, and free trials of soapUI pro, loadUI and AlertSite, are available at <http://smartbear.com>



Ride the wave.
Stay on top with TestWave.

Testing is complex - why not make it simple?

Running a test project takes coordination, patience and endurance. You need to keep track of what's been tested, what needs to be tested and who should be doing it. Without a test management tool, your work just gets harder. TestWave keeps track of all your test cases, requirements, releases and defects in a central location to improve your efficiency. Since TestWave is cloud based you can be up and running in minutes, not weeks, allowing you to be immediately productive.

- A full test management tool incorporating requirements, test planning, execution and defect management
- Delivered and hosted online: no complex installation and no costly servers (onsite option also available)
- Test teams can be using TestWave within minutes from anywhere in the world
- Extensible - interfaces with applications such as JIRA® and Quick Test Pro®

For a free 30-day trial or for more information visit our web site.



TESTWAVE
www.testwave.co.uk

TestOps

by Edward Bishop

Testing still has no clear role in agile, but is central to DevOps



PT editor
Edward Bishop
envisages automated
operations testing
using Ranorex

Many PT readers work in environments where a new release of the product is an important event which, to avoid severe loss, must take place on a date arranged long before. That may be for example because users require or have been guaranteed new or changed features from that date, to maintain business-critical compatibility with other systems, or to comply with changed regulations. In the case of publishing user-distributed (“shrinkwrap” or “commercial off-the-shelf”) software it might be because facilities have been booked to produce, finish and distribute physical media. The replacement of that by online distribution helps by reducing lead times but rarely makes the deadline, chosen for strategic or set by contractual reasons, more flexible. It hardly ever makes the quality of the release less important because asking users to apply urgent updates soon afterwards will usually cause business damage of the same kind as lateness of the main release.

DevOps, the concept that developers and operators (system administrators, infrastructure engineers, DBAs etc) should

work more closely together, applies to none of these situations. It has evolved from experiences gained in “continuous” environments, where the application – often public-facing – remains centralized and is accessed by users via web or cloud. That has the huge advantage to those producing it that it can be changed quickly and easily.

The resulting shift in priorities should be an opportunity for business improvement. It empowers those making decisions about the release to take into account more factors, especially the expected behaviour of users and customers. It makes their choice less stark by providing an extra option: release parts now and more later. It makes it possible to perform a balanced, strategic release based on business optimization (usually revenue maximization) rather than technical risk.

DevOps is often confused with agile development. The latter is older and can be used in the situations described in my first paragraph (which is not to say doing so is necessarily advisable). It was invented by developers frustrated by working to requirements who felt that maintaining documentation and resolving incidents raised because of apparent differences between it, its sources and what they were producing made them less creative and productive. The most enjoyable part of programming is solving problems, therefore a programmer allowed to write wrong code then fix it will tend to be happier than one disciplined by processes designed to minimize, rather than repair, defects. The code-fix approach requires frequent builds with rapid empirical testing to provide the information needed to fix. The logical extension of this is continuous integration, where the build being tested is always up to date.

Thus developers, as they often have and do, won the right to work the way they

prefer. Years later, whether or not that is a good thing is still an open question and the argument shows no sign of being resolved soon. On one hand, agile necessarily wastes work: code, and therefore tests, are repeatedly replaced and revised. On the other, agile can make accommodating frequent change, especially of requirements, faster and easier. Much depends on roles: agile may be appropriate in situations where developers define requirements and bear cost and risk. When those responsibilities rest with others, questions about how their interests can be adequately protected become more complicated. The developers who first advocated agile, and many of those doing so today, had and have no understanding of nor interest in independent testing. Indeed, ignoring testing is one of agile's key aims: the developers do not like being disrupted by the availability of information indicating that work done so far should be corrected before further work is done so they have done away with any adequately formal definition of correctness, making testing impossible. To make any contribution testing has had to change radically, discarding many of its proven best techniques. It is still trying to discover how to make that change and testers in agile environments are either marginalized or, more often, not really testers but a kind of developer or developer's assistant. "Test-driven development" is in no way a replacement for testing because it compares code with its own component-level design specification, such as it may be defined, rather than against requirements.

Continuous integration, a preferred method of some developers working on builds, is quite different from continuous deployment, the *raison d'être* of operators responsible for releases and the configuration and data structure changes they require. It is not acceptable intentionally to risk causing failure when that will cause risk and loss in real time. Operators must aim to minimize incorrect behaviour of all kinds, functional and non-functional, and that's incompatible with agile's try-fix-try again approach. This confusion – the idea that the point of DevOps is to release as often

as you build – prevents the real advantages of releasing more frequently from being realized. The role of independent testing here is to qualify, or not, candidate builds for release, enabling decision makers to choose the most effective and least risky of those available to deploy at any given time.

Releasing even a qualified build often causes unexpected effects due to differences between the production and pre-production environments. Making them as similar as possible helps, and innovation in virtual infrastructure is reducing the cost and effort required to do that, but the problem is far from eliminated. For the foreseeable future operators will continue to need to regression test after every change, immediately and as quickly as possible, in order to decide whether to back the change out in order to minimize the number of users that experience failure or, even more importantly, avoid disastrous data corruption. The role of testing is to provide the means to perform the necessary testing effectively, based upon the current state of fast-changing business requirements rather than upon the expectations of developers.

To some, "DevOps" means something else: that two roles should be merged to create a new one. This is already reality in small

web teams, and cloud is making non-web more like web. The devop needs the skills and knowledge of both developer and operator. He or she has sufficient knowledge of the code to be able to modify it to fit operational requirements. *How will these modifications be tested?* Rather than just changing configuration or data, the devop is responsible for writing code to perform, reverse, amend and manage that change, ready for reuse and perhaps integration with the product. *Will this dangerous code be independently tested?*

If merging roles with different goals and mindsets in this way seems far fetched, consider this: agile has already done it for testers and developers. Agile development teams with responsibility to anyone other than themselves always contain at least one person we might call a "testdev". It has been suggested by some agile advocates that every team member should be able to do any other member's job. This false ideal ignores human nature and the fact that the longer one remains in a specialism, the better one can become at performing it. Nevertheless, if operators are to become devops, testing must help them become testdevops.

Ranorex and TestDevOps

I described previously in PT how the code-level approach used by the test automation

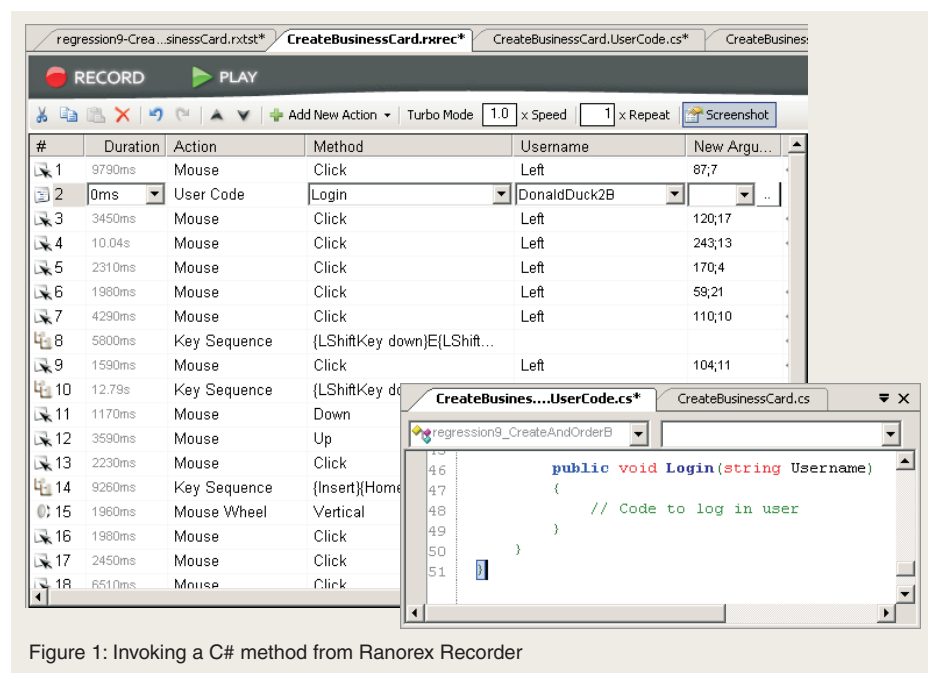


Figure 1: Invoking a C# method from Ranorex Recorder

tool Ranorex facilitates powerful sharing of test assets between testers and developers (<http://www.professionaltester.com/files/PT-issue8.pdf>) and enables innovation in test design and implementation (<http://professionaltester.com/files/PT-issue13.pdf>). The same facilities offer approaches to fulfilling the roles of testing in DevOps I've identified in this article.

Ranorex Recorder, the module used to automate test procedures, is extensible. Because the recorded procedure is simply a code module, it can invoke and pass parameters to methods in a C# class (see figure 1). These invocations, added during or after recording, create great potential for test code reuse and for integrating testing with ops. For example, by writing simple classes called login() and logout(), the testdevop can be provided with parameterized suites executable against any easily-specified group of test, or real, users, without access to their passwords or necessity for dangerous backdoor passwords.

A Ranorex project is a .NET project, so any test suite can be run by launching a .EXE file from the command line. Doing this manually is fast and easy, which is important to operators who need to execute selected suites repeatedly after each step in the change they are carrying out, including backout steps. It also makes it easy to port tests to multiple real or virtual clients for simultaneous execution of different tests against the live system, saving crucial time. More importantly, it enables the testdevop to integrate real testing into the batches and scripts used to automate operations. A script can be written that tests the effects on users of its own actions and, if they caused regression, backs them out.

Ranorex identifies an interface object by its "RanoreXPath", similar to an XPath with the client OS GUI as the root. This means test suites can include control of and validation of the output of any combination of applications including ops and monitoring tools.

Thus it becomes feasible to automate validation of automated operations steps from the user's, operator's and application manager's points of view. This is how the dangerous concept "control everything in code" proposed by agilephile DevOps advocates can be made safe and workable ■

A free trial of Ranorex is available from <http://ranorex.com>

DevOps, ALM, Process Improvement and Agile Testing events

Professional Tester magazine is the Media Sponsor of four conferences taking place in Amsterdam and London during November and December.

The series begins with three co-located events in Amsterdam on 15 November which address the specific themes **DevOps**, **Application Lifecycle Management** and **Process Improvement**.

DevOps Summit Europe: Enabling DevOps

15 November, Amsterdam

www.devopssummit.com

Application Lifecycle Management (ALM)

15 November, Amsterdam

www.unicom.co.uk/almforum

Recent Trends in Process Improvement: Focus on Products & Services

15 November, Amsterdam

www.unicom.co.uk/processimprovement

The 8th Next Generation Testing Conference:

Agile and Business Focused Testing

6 December 2012, London

www.next-generation-testing.com

For information:

+44 (0) 1895 256 484 info@unicom.co.uk






5TH ANNIVERSARY

One of the Biggest Conferences
on Software Quality and Testing

Jan. 15 - 17, Vienna

Quality - Investment for the Future

Keynotes, practical presentations, exhibition, tutorials,
solution provider forum, scientific track,
tool challenge, networking,  **STEVE FACHTAGUNG**

TOP SUBJECTS:

- Requirements
- Testing and automation
- Quality management
- Embedded systems

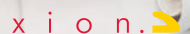
TOP KEYNOTE SPEAKER:

- Prof. Manfred Broy
- Richard Mark Soley
- Hermann Scherer

➤ **Register online now!**

www.software-quality-days.com

**MORE
INFO
ONLINE!**



11th International Conference on Software QA and Testing on Embedded Systems

This year we offer a special program: **3 Keynotes, 2 Tutorials, 8 Tracks**, from renowned companies such as: **Intel, Samsung, Phillips, Adobe, Oracle and Quality Perspectives.**

QA&TEST, the international Conference on Software QA and Testing on Embedded Systems, will be held on 17, 18 and 19 October in Bilbao, Spain, and gives you an excellent opportunity to increase your business network and establish contact with others professionals of the industry.

The main objective of QA&TEST is present the last technological developments in Software Testing and Quality Assurance, and showcase successful best practice to reduce costs and give companies a lead in global competition.

www.qatest.org

Special offer 15% Discount for Professional Tester Readers (Promotion code PROT2012)

Organiser:  Sponsor: 

October 17 · 18 · 19
2012
Bilbao · Spain

**Free
30-Day Trial**

Now Available at
www.adminitrack.com

**Web-Based
Issue Tracking**
Effective and
Easy to Use

"Find out why
project teams
worldwide
prefer
AdminiTrack
for their Issue
and Defect
Tracking needs"



AdminiTrack.com

BizTestDevOps

by Wolfgang Platz

To shift left, specify change directly as business test cases



Wolfgang Platz on testing as a means to govern DevOps

Whatever DevOps changes, testing must continue to govern development. Testing in itself is not a changing force: rather, business is. Business changes drive the need for higher quality testing with less risk. The effect of moving development and operations closer together is to broaden the role of testing by making it necessary for testing also to govern operations.

Governance of development requires early test specification

To meet its aim of optimizing the efficiency of the process by which business requirements are fulfilled, testing must begin before development (including development of change to an existing product). Testing has always striven, correctly, to do more earlier, but that is often prevented because business and development perceive it as a brake, preventing progress from building initial momentum. Experience has caused business to become more enlightened, as expressed in the now often-expressed desire to “shift left”, that is to place more hard and risky work earlier in the timeline where, if it does not go as hoped, it leads

to less damage with more time and options for repair. That won't be achieved by earlier testing alone, but can be by testing that begins to drive and inform development earlier. In my opinion the way to do that is early production of comprehensive test cases which form an integral part of the requirements specification. This concept is sometimes called “specification by example”.

Governance of operations requires early test automation

The information contained in the test cases is also critical to the handover of the new or changed product from development to operations. Where infrastructure considerations allow, governance may be applied by disallowing change in production until it is shown that all test cases pass with the change in place in pre-production. More typically, governance means putting decisions about the change into the hands of business management, not developers or operators. The test cases do that by providing correct, current, business-interpretable information on the remaining business and technical risk.

Neither governance method is possible unless the test cases can be executed very quickly and efficiently. Given that, testing achieves “business shift-left” a second time. A high level of test automation empowers governance of operations, working to prevent failure too close to, or in, production. To achieve that, *the test case portfolio must be not only defined early but automated early* which necessitates deriving the automation from the requirements, not having to wait for an executable build of the product to do so.

The need for speed

People working in agile development environments are kept short of time deliberately. It is by this self-enforced

urgency that the process is driven forward in an attempt to ensure it does not fall behind the real, external, business urgency that requires the product. However attempting to shortcut requirements specification is a false time economy. When deriving and refining requirements in discussion with users, teams must be able to record exactly what is required. Otherwise, they have not understood the user story and do not know what is required. This will lead to over-fulfillment, the development of what is not required or necessary. The measures taken to prevent that must be extensive.

The question is how to record. User stories are never complete nor completely explicit, yet assumptions and misunderstandings cause waste and delay. So the answer is not glib snippets written on sticky notes or cards. Neither is it natural language documents which take great effort to produce and are never right, or models which are open to interpretation. Skip all of these and go straight to concrete specifications by example, ie test cases. With these, developers know always exactly what they are tasked to develop against. They are like code: they may vary in elegance and efficiency, but they are either, and provably, correct or not.

Agile is a rapid development strategy therefore requires a rapid test case definition strategy. At TRICENTIS we typically spend two hours designing test cases to describe a very long and complex user story using TOSCA Testsuite. That's nothing compared to the effort that will follow, but reduces that effort and the risk it entails because when it is done everyone knows the specification is correct, the user story is properly understood and the initial design for the handover to operations is ready.

Governance and coverage

In TOSCA's approach to test case design and automation, "coverage" refers to measurement of business risk. It means being able to state accurately at any time the business risk of putting the test item into production at that time. TOSCA achieves this

because its starting point is a risk-based structure of functionality down to the level of user stories, so we know exactly the business value represented by each test case, the risk if it fails, and the need to design and/or execute more test cases.

For example, the risk can be described to decision makers thus:

- we have proved that functionality representing 95% of the product's business value (ie business risk if functionality were not available) will work correctly
- we know that functionality representing 1% of the product's business value will exhibit specific, known failures we can describe exactly. You can choose to live with them or wait for repair and retest
- the remaining 4% of business risk has not been covered: we don't know if or how it may fail. You can choose to take the risk or pursue further testing.

This is far more meaningful than talking about numbers of tests executed, passed and failed. Without knowing the coverage and therefore relevance of each test case, that tells those responsible for governance nothing of use.

Governance of multi-platform development

DevOps has evolved in part because of the diversification of the work of software organizations, with multiple versions of target applications being delivered for and using multiple platforms, devices, technologies and operating systems.

In governing all these different developments and operations, carried out by different people, the critical success factor is that there is always an interface to the functionality to be tested, presented from the testing perspective on the most elementary level possible.

Take for example a native app (or rather, three apps) for three different device OSs. Both client components interface to a

server component that performs business logic. If the component also serves a web application, this interface is probably already described by a web services specification using one of the many available standards. If it is not, development must be required to ensure that all functionality of the server component can be addressed via a public interface of this or other types, for testing purposes.

Using that interface, many if not most of the important test cases can be executed with no need to involve the client device. A few simple additional tests are added to test the connection of the device to the interface. Now it is simple to mirror those additional tests for other devices.

This vertical decoupling (that is, converting dependencies into services) of layers can and should be applied to multiple layers of other systems too. It does however have one weakness: the interface specification is likely to be technically complex, making it hard to relate business requirements to the tests needed to assure them. TOSCA resolves this issue using its OneView technology which presents any test case specification, regardless of whether it is for manual or automated execution and relates to a GUI or non-user-interface, on the business level in an easy to understand format. This is the key to increasing the amount of testing which can be automated early, with all the attendant advantages I have already described.

In some cases more testing of the client component is needed because it, rather than the server component, performs significant business logic. The first target for early test automation should still be non-GUI methods: these are always more (typically, I estimate, around five times more) stable than GUIs. TOSCA's instruction layer abstracts a test case from the specific interactions with the GUI as designed for a given device needed to execute it. Having the knowledge required to execute the test (called "steering information") is delegated to a manual tester at first, then captured and automated when the differences between them and

the equivalent tests on another device have been established. Hence there is only one set of test cases, purely business-driven, for all devices; only the steering information differs.

Big data and compliance with external governance

Many TRICENTIS customers in the financial and banking sectors need to provide comprehensive risk reports to various external authorities, for example for compliance with the Sarbanes–Oxley Act or the eighth EU Company Law Directive (sometimes called “EuroSOX”). Failure to do this to an exacting standard of accuracy can lead to exclusion from markets with enormous financial loss. The systems that create these reports therefore require rigorous testing, of their data processing functionality but also of the quality of the initial input data they derive from the primary systems.

It is common to do this latter part of the testing manually, using hypercomplex SQL queries executed against the different staging levels. The complexity makes the work error-prone and the very large volumes of data and the high resource consumption of performing, especially, operations using the JOIN keyword cause long processing times, limiting the checks that can be performed.

TRICENTIS TOSCA@BigData provides the advantages of OneView using new and unique technology created specifically to address this issue. Again the starting point is business-driven test cases defining what needs to be done. From these TOSCA creates, dynamically at runtime, an optimized set of SQL queries containing as few JOINS as possible, and requiring each of these to be executed only once. Advanced aggregation and examination algorithms allow comprehensive checking of the data returned.

Self governance

No discussion of testing as a means to governance would be complete without defining a means to validate testing, that is

to provide evidence that the testing being done is a good means to governance, in other words to measure the effectiveness of testing. In my opinion the most important metric for this purpose is the weighted good/bad defect ratio.

A good defect is one that is detected in time to prevent it reaching production. Whether that detection happens during early test design or in pre-production with minutes to spare makes no difference. A bad defect is one that reaches production so that a user can become aware of its existence. Whether a defect detected in UAT is good or bad depends on who detects it, but most TRICENTIS customers would always consider it bad because (i) user representatives *could* become aware of it and (ii) it disrupts and delays UAT, necessitating a wait for repair followed by a return to system testing for retesting and regression testing. The

weighting is simply the severity currently assigned to the defect in the incident management system.

This metric does not take into account defects that have not yet been detected. In my view, it should always be assumed that no more defects will be detected: in other words, a defect not yet detected should be considered not to exist. This may seem odd, but it is the only logical course because including guesses about imagined defects that may or may not be found in what is supposed to be an empirical metric is wrong. For that reason, metrics like this are purely retrospective. They indicate past, not current product quality and are in no way a guide to remaining risk. The information needed for governance, that is predictions about defects that may be yet to be found, can only be obtained using detailed, granular coverage information ■

Wolfgang Platz is founder and CEO of TRICENTIS. For more information about TOSCA Testsuite visit <http://tricentis.com>



The Test Data Generator

GENERATES ANY KIND OF DATA FOR YOUR TEST

Q-up allows time and cost savings up to 90% compared to manual creation of test data.

- Create meaningful test data
- Specify business process oriented test data
- Automatically generate test data
- Connectivity and performance

Have any questions or need detailed advice? Feel free to call us:

+49 6171 69410 - 0

A product of



Obere Zeil 2
61440 Oberursel
support@q-up-data.com

WWW.Q-UP-DATA.COM

**PROFESSIONAL
TESTER**

Essential for software testers

Tell a friend

**SUBSCRIBE
It's FREE
for testers**

professionaltester.com

Quality Level Management: who, what and how

A new model for managing IT product quality in outsourcing relationships



Our series of features on how testing services should work continues. With **Jeanne Hofmans** and **Erwin Pasmans** of Improve Quality Services

In the first edition of LITS (see <http://professionaltester.com/files/PT-issue16.pdf>) Vinoth Kumar described the information Cognizant provides to users of its testing services. That is a critical activity in any service model, but especially that of a testing service because measuring the effectiveness of testing is very difficult.

Testing should increase confidence in quality. That does not happen because few defects are detected, nor because many are detected and fixed: both phenomena should decrease confidence. Increased confidence comes from knowledge and understanding of the passed tests, gained by digesting much information. It's not easy, but it can be achieved by close observation of the testing work as it proceeds, or ideally being involved in it personally. Converting testing into a service removes first-hand visibility and creates dependency on reporting alone. A test report is by definition a summary, that is it deliberately omits information: it would be impossible to use otherwise. To have accurate knowledge

of the quality of the product the service user must have complete understanding of the meaning of the report. That requires complete knowledge of how it is created, ie what is measured and how, and how it is summarized.

Jeanne Hofmans and Erwin Pasmans are currently completing their book on quality management in outsourced projects with large IT components. Here we ask for their advice on how to look inside testing services.

Who should engage the testing service?

Reports must be designed for the reader. Cognizant's are for "business and IT stakeholders". Other roles, for example project managers, lead developers or test analysts, would require very different reports in order to be able to achieve confidence in the testing done and therefore in the product.

In your opinion, which role is the easiest to which to report well?

JEANNE HOFMANS and ERWIN PASMANS:

Quality is a subjective concept. In his famous book *Quality Software Management: Systems Thinking* (Dorset House, ISBN 9780932633729) Gerald M. Weinberg defines it as follows: "quality is value to some person". James Bach modified this to "quality is value to some person that matters", making explicit an additional point already made by Weinberg in the original context. That subjective nature makes defining and establishing quantitative quality reports very difficult, if not impossible. Thus the question "which role is easiest to which to report well?" should be replaced with "who matters most?". Once that is determined it should be determined what matters most to that person or persons. In other words who is easy to report to is not relevant. You

should report to the people who matter and report on what matters to them. They have probably had a hard time defining what matters most to them and will probably change their mind over time.

That is why we agree that increased confidence comes from close (personal) observation of the testing work as it proceeds. We disagree however that converting testing into a service removes first-hand visibility and creates dependency on reporting alone. To be successful one should not depend on reporting alone. The visibility should be stimulated and simulated (eg using cameras and screens) as much as possible. Visibility is a key factor in the success of metrics. They should be shared amongst the team. Preferably both customer and supplier are able to view the metrics in a shared dashboard. Using this dashboard as an entry point, team members such as lead developers and test analysts find the detailed information that is needed. This visibility and openness is not only applicable to the metrics of the product, but also to dashboards that report on process level and to the organization as a whole. On organizational level it helps to pay visits so that team members get to know each other. It also helps to have screens in the office displaying the team working at another place.

The focus on several levels (product, process and organisation) is key in our book, in which Improve Quality Services presents a new model that addresses several solutions of managing quality in outsourcing. Numerous solutions are already widely available for problems in either outsourcing or quality management, but until now there was not one universal model or framework to approach these problems.

The Quality Level Management-model (see figure 1) has two main dimensions regarding measures to improve and sustain quality: *levels* at which measures can be taken: organization, process and product; *types* of measures: preventive, detective and corrective.

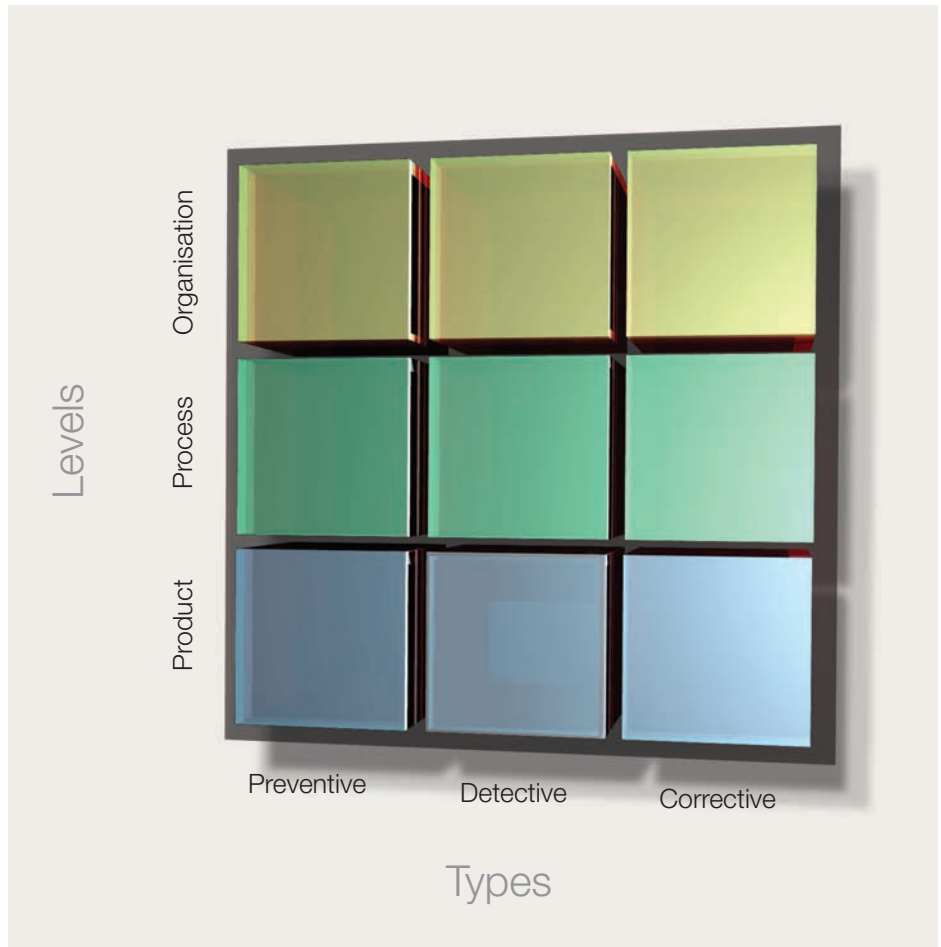


Figure 1: levels and types of measures in the QLM-model

Collecting metrics is an example of a detective measure on product level. The use of dashboards, showing the progress of testing is an example of a detective measure on process level. Determining relevant stakeholders and involving them is an important activity on organizational level. All levels are needed to report meaningfully about quality.

What should the service provider measure?

Different reports require different inputs.

In order to report meaningfully to “who matters most”, what needs to be measured and how should that be done?

HOFMANS and PASMANS: To report meaningfully a key factor for success is to limit the amount of metrics. A limited amount of metrics is easier to understand to all involved, especially because metrics should be interpreted carefully. Few defects

does not automatically mean the product is of good quality. It could also indicate testing is not taken seriously. The effectiveness of testing decreases as it takes much more time to decently report bugs. Is a testing service performing badly if the code that is being delivered is poorly maintainable? Perhaps users are satisfied with the product delivered but the maintenance department is not. And is that same testing service doing a good job if they achieve 100% requirement coverage? Perhaps the requirements are poor or very generic. Perhaps every requirement is traceable to a test case, but that test case covers only a small part of that requirement. Just measuring test effectiveness is not enough.

As in test framing, the process and the story of the product must also be told. The story contains the highlights of the test approach, the constraints of the test process and the results of testing. Often the customer is not specialized in IT processes. The story of the product contains more information than a report full of metrics. 80% decision coverage is meaningless if the code of the remaining 20% is used most often or in critical parts of the product. That is why reporting should be about risks. This can be accomplished by listing items covered versus items not yet covered by successfully executed test cases. Not just the reporting, but also the testing itself should be about risks and defining appropriate measures. Stakeholders worry most about potential failures and their impact. Therefore these risks should be agreed upon by the stakeholders and reported back during and after testing.

How should the service's reports be actioned?

Testing should aid decision making.

How can who matters most know (i) the risk of allowing a test item to pass out of their responsibility; (ii) by how much and how quickly more testing can reduce that risk?

HOFMANS and PASMANS: An important aspect is that good testers are aware of defect

clustering, the phenomenon that defects tend to cluster in one area or component. A good tester does report this so it can be decided whether further targeted testing should be performed. Good testers also divert from the test script if there is a reason to do so. They also report strange side effects: dynamic implicit testing. This is impossible to grasp in quantitative metrics. The use of both quantitative metrics and qualitative information of testers aids decision making based on risks.

The best chance for success in passing the responsibility of a test item is to use detective measures at all three levels. At product level review the test cases and requirements, perform some witness or acceptance testing. At process level perform some collaborative quality scans or audits to check if risk analysis meetings are held, if configuration management is working properly. At organizational level it is very wise to talk to the people involved. Knowing that testers are well capable and understand the perceived risks of

the stakeholder can give a huge confidence boost in the testing performed.

Reducing risk is not just about taking detective measures like testing but more importantly about taking preventive and corrective measures. These preventive and corrective measures are applicable to the organizational, process and product level as well. The QLM-model describes all these types of measures. A preventive measure on organizational level is to achieve a level of trust and confidence between customer (stakeholder) and supplier. Because trust alone is not sufficient measures at process level and product level are needed, for example an incident management process that is easy to use by both partners or the use of coding standards. Quality is not achieved by just testing but also by good design and development practices. Our book on the QLM-model is therefore not just about testing services but covers all aspects of managing quality in outsourcing: quality level management ■

Jeanne Hofmans and Erwin Pasmans are test consultants at Improve Quality Services (<http://improveqs.nl>). Their book Quality Level Management: Managing Quality in Outsourcing, will be presented at Eurostar 2012

Structured testing



Testing with proven methods and techniques

Quality Level Management



Managing quality in a customer – supplier relationship

Agile



Testing and teamwork in innovative development processes

Improve Quality Services BV

Laan van Diepenvoorde 1
5582 LA Waalre

Amsterdamsestraatweg 55a
3744 MA Baarn

Phone +31 40 202 1803
info@improveqs.nl
www.improveqs.nl



S.E.A.L.*

The perfect solution to cut in your testing costs

1. CONFIGURATOR

define your test data environment

2. SELECTOR

select the test data you need

6. COMPARATOR

view all differences before and after test

3. EXTRACTOR

based on your selection, S.E.A.L. extracts a consistent data set

5. ANONYMIZER

protect sensitive data

4. GENERATOR

add extra data when needed



Come and visit us

at Amsterdam RAI on booth nr 50 between 5-8 november:
www.eurostarconferences.com or contact us at www.rever.eu



* S.E.A.L. = Select, Extract, Anonymize, Load

Put the crystal ball away

by Martin Mudge

You find out whether more testing was needed only after it's done



Martin Mudge suggests a new metric which is really a criterion

Measuring the effectiveness of testing is a favourite topic in PT. Good ways to do it are desired for several reasons: to help decide when to stop testing, to inform process improvement, and to help sell or justify testing to those who commission it. The increasing trend to convert testing into a service, discussed in detail in the last issue (see <http://professionaltester.com/magazine/backissue/PT016>), has made the latter reason especially prominent. Thinking about the information a consumer of such a service needs to make decisions about it led me to consider the cost of detecting defects, a metric I could not find mentioned anywhere.

The cost of nondetection

It is common to ask consumers of testing – as a service or not – to consider *the cost of not detecting defects*. In other words, we take a known defect and try to predict what might have been the business impact had it entered production. In some cases this is done quite easily. In others, it depends on unknown factors, for example which user or users would have been affected by it first and how he/she/they would have reacted.

A more fundamental weakness of this method is that severe defects can wreck projects before production as well as products after, and the earlier the defect is found the harder the speculation becomes. For example, suppose a review detects a critical ambiguity in a requirements or design document. If you want to, as testers always should and will, you can argue that left unfixed this would have caused incorrect development which would then propagate itself like wildfire until the product was so riddled with defects that it would be cheaper to start again than try to repair it. On the other hand empirical testers and developers will claim that they would have found the defects the ambiguity caused by other means a little later with no great harm done. The truth is no-one knows.

The cost of detection

It is much easier to calculate the cost of finding the ambiguity. We can measure the time spent conducting the review by people whose cost-per-hour is known, thus calculate the cost of each defect it detected. That simple exercise will usually provide a powerful argument for early lifecycle testing and the production of the documentation it requires.

However in this case things become more complicated the later a defect is found, because it is not obvious where the cost lies. Suppose executing a test suite takes t hours and detects 2 new severe regression defects. The cost per defect is not necessarily derivable from $t/2$ because that does not account for the previous work required to build and maintain the test suite. It also raises a difficult question: if the test suite detects no defects does that mean it has no value? Clearly not: its value comes from its potential to detect defects should they exist. But its value is diminished by its potential to miss defects.

Not knowing whether or not they exist nor what they might be, we cannot know what is that potential. Wolfgang Platz, in this issue of PT, recommends a compromise: take the ratio of detected to missed defects (both weighted by severity). That is probably a fairer way to evaluate the effectiveness of testing but, as Platz also points out, that evaluation is only of testing already done. Decision makers may choose to use it as a guide, but hopefully are aware that despite what macro-historians may say, the recent past is in no way a guide to the imminent future.

The value of detection

Because of these difficulties in considering the cost of detecting a defect as a metric and trying to measure it, it is better to consider it a criterion to be specified. Obviously the correct entity to do that is business.

Many testers, including the editor of this magazine, argue that testing to assure the absence of defects is just as valuable as detecting defects. I disagree, and share Cem Kaner's view that "the primary function of the test group is to find bugs, and the primary work product of the individual tester is the bug report". The title of the article in which he expresses this is *Don't Use Bug Counts to Measure Testers* (http://techwell.com/sites/default/files/articles/SmzXDD2217filelistfilename1_0.pdf) and doing that is certainly not my aim here. But I do think that the value of testing to a business is a simple function of the number and severity of defects it finds. If analysts and developers do so excellent a job that there are few defects to find (extremely unusual), that is not testing's fault: but we should be realistic and accept that it does make testing less imperative.

Typically testing makes a proposition to business along the lines "this much risk remains. Do you want to accept it, or should we continue to attempt to reduce it at this cost per day?". This is a very tough decision because risk is of the future. Reducing risk does not mean detecting defects. It may mean trying but failing to

detect defects. If that happens the cost, in retrospect, could have been saved. Trying to do that will be the natural instinct of many to whom the question is put.

So I suggest another proposition format: "would you be prepared to pay, and if so how much, for detection of a defect of this type, where the type is defined by you in terms of the impact it could have on your business?". It seems to me that this is a much easier question to answer and that, from the point of view of a tester wanting to test, the answer is more likely to be positive. That's because the proposal is about buying value, not being forced under threat of disaster to accept cost.

The you-pay-as-we-detect model

The reason for choosing to use a shared testing service, whether organizational or external, should not be because one is prepared to tolerate less effective or less comprehensive testing. In fact I believe that a testing service can and should surpass the performance of the internal testing it replaces. To me, the important selling point of services is their flexibility. When confidence in the product increases, service consumption and therefore cost can be reduced immediately. The opposite is also true. The onus on management to try and predict these unpredictable fluctuations long in advance so as to avoid

expensive under- or over-resourcing is removed. In short, using on-demand services makes testing more agile.

The best factory models adjust themselves in this way transparently. For example, at BugFinders we charge per defect found. That charge varies with the type of defect, according to typing criteria agreed with our customer beforehand: for example, many customers distinguish between "GUI" (presentational, eg misspelled word or missing non-critical image) and "functional" (user cannot complete required business action) defects. Our customer also specifies a maximum spend. Obviously we want to reach that, by detecting as many defects as possible and prioritizing detection of high-value defects. When the detection rate is high, we allocate more testers (who are also paid per defect detected) in order to do that more quickly. When it falls, we adjust the approaches and techniques we apply in order to try and increase it, according to our judgement on how to maximize our revenue, which is exactly the same thing as maximizing the value, as defined by our customer, we provide. The customer always receives the best and most appropriate testing in the current circumstances, at known maximum cost, with no need to manage or resource it ■

Martin Mudge is the founder of BugFinders.com. His white paper How Much Do Your Bugs Cost? expands on the concept of this article and is available free at <http://bugfinders.com/training/downloads/how-much-do-your-bugs-cost>

Trending: testing



Submit VM images with incident reports

The Windows VM provided to me by the powers that be takes about 19GB of storage. Not small, but not unmanageably big. So before I start testing (manual or automated), I make a copy of it. When I observe an anomaly, I immediately suspend the VM and add the date, time and perhaps a short note about the defect to the filename of its image. Then, depending on my judgement of which is best, I either continue testing in the clean copy (remembering to make a copy of it again first) or make a copy of the one I just suspended and carry on working in that.

If development says it cannot reproduce an incident, I provide it with a copy of the image of the VM at the time I observed it. The VM, once running, provides all the configuration and state information devops need. They can even use the undo function to discover my user actions prior to the incident.

Could this method spell the end of arguments about reproducibility of incidents and therefore of irreproducible incidents?



Why risk does not work

Imagine five people are at sea in a lifeboat. It's holed and sinking fast. You are piloting a rescue helicopter overhead. There is no chance of any more help arriving in time. You have only two options.

Option 1: use your winch to take three of the people on board the helicopter. It can't fly with any more. The three winched up will definitely be safe. The two left in the lifeboat will definitely die.

Option 2: try to use the helicopter's downdraught to blow the boat to shore. If you succeed all five people in the lifeboat will definitely be safe, but according to trusted statistics the likelihood of the boat overturning is exactly 0.5. If that happens, all five people in the lifeboat will definitely die.

You can't combine the options.

Option 2 requires flying at an angle and is impossible with anyone on board except the pilot. None of the people in the boat can fly the helicopter. You must choose either option 1 or option 2.

Which option will you choose? Why?



Difference between optimist and pessimist

The pessimist knows all the facts



Difference between RAD and agile

In RAD, developers make it up as they go along and do whatever the hell they like. Agile is subtly different: developers do whatever the hell they like and make it up as they go along



Appalling failures caused by IT suppliers

Fujitsu has reportedly (<http://www.ft.com/cms/s/0/0d1595d6-fb6f-11e1-b5d0-00144feabdc0.html#axzz26AkAAH5D> you have to register to read it so we suggest you don't bother) been blacklisted by the UK Government at the behest of the famously competent Francis "jerrycan" Maude. Our question is: why have all the other oligarchy IT suppliers to UK Government *not* been blacklisted?

PT reads everything about testing on the web so you don't have to.

For the latest thinking that matters on testing visit professionaltester.com

To tell us something else please email editor@professionaltester.com

Amazing Ops through Test Automation

Testing
Technologies



Never mind all the stories you have heard about the nightmare of test automation. Your test automation project does NOT have to fail because you are short on people to develop and/or maintain your test system, or the costs are exploding when developing your test system, or you can not justify the maintenance costs of your test environment to your management, or...

Testing Technologies has advised and supported many successful projects on test automation. Take this one for example: A small team of a Belgian operator for the radio communication network managed to automate their challenging testing needs in a relatively short time, right on schedule, right on budget.

But how do successful projects differ from failed ones?

The first step is to consider some practical views. A test automation project typically never starts on a green field. In most cases, there are already tools available that have reached their end of life cycle, and/or there is a collection of different test tools that have been developed in-house. Projects driven by software developers usually come up with a pretty enlarged tool landscape which is, however, lacking the support of 'real' testing challenges. Overseeing basic features (from a test perspective) makes it one day virtually impossible to maintain or enhance the existing test tools any further. Testing tools created in projects driven by test engineers come up pretty well in respect of testing issues but show shortcomings from a software design perspective.

Many businesses decide to purchase or license a ready to use test tool. But even though test automation software companies spend lots of engineering money to develop optimal testing tools, they are usually only efficient in the focused domain, performing poorly when applied outside the intended area. The only way out of this situation is to use

a test technology well designed by testers for testers with a solid test system architecture designed by software developers for software developers. To perfect this, get some commercial tool and service support that is backed by a strong community in your particular domain.

And the Belgian operator found just that!

Scouring the existing commercial tool market, they took notice of Testing Technologies' TWorkbench, which had already been applied in their particular domain. Its unique features, especially how test progress is visualized and reported, convinced them to give it a try. But no off-the-shell tool support had been available for their particular problem. Not yet! The extension capabilities of TWorkbench via open and standardized APIs enable the implementation of additional functionalities. Testing Technologies provided all missing features as an off-the-shell solution just in time, so the operator's testers could immediately start to create their specific test environment. To implement the same functionality would have been an alternative option.

As a result, the Belgian service provider not only deployed a highly customized but off-the-shell test environment, also all their investments in building this particular test infrastructure can be reused in the future too, as the integrated TWorkbench is based on an internationally standardized test technology called TTCN-3. This technology was created by testers for testers in an international, technology-independent environment, and was implemented by software developers for software developers.

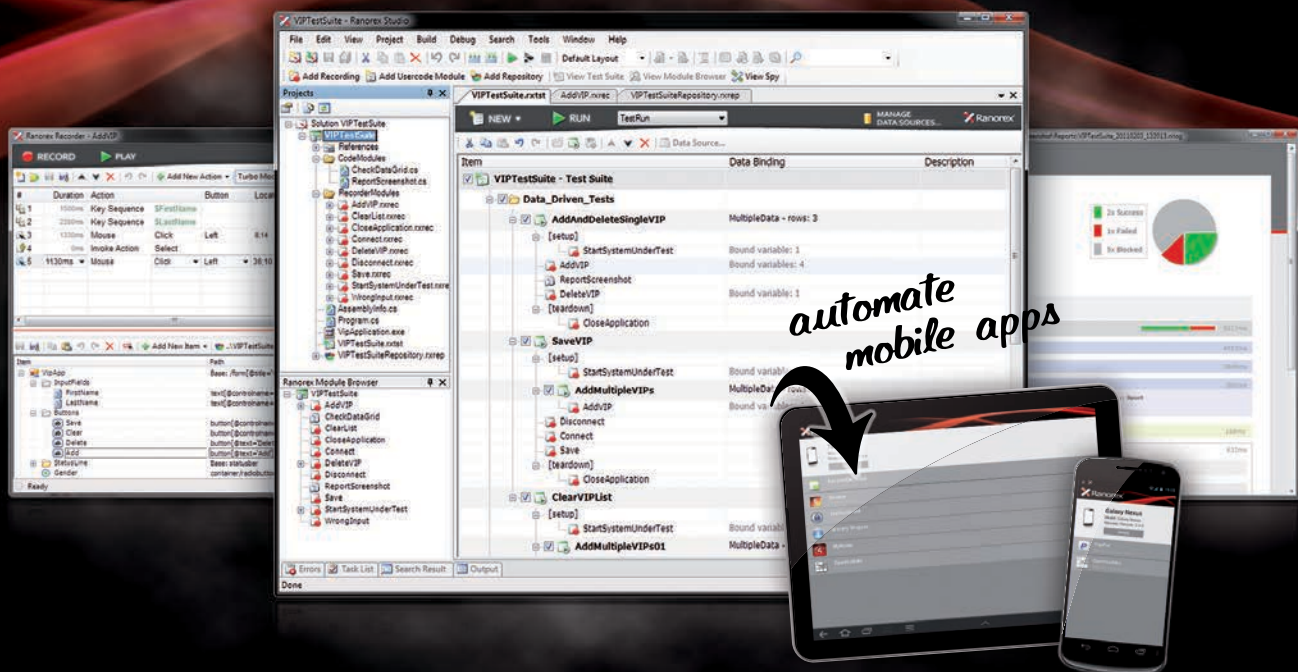
Don't overlook your amazing ops through test automation! Read the full story at www.testingtech.com/testdevops and spare further nightmares with our support.



Enjoy your extra time! It can be so easy to create your test automation solution.



Automate Testing of any Windows, Web & Mobile App



- ✓ Use connectors for data-driven tests
- ✓ Build robust test automation frameworks

- ✓ Generate EXEs for pure flexibility
- ✓ Write custom code in C# or VB.NET



*Record and Edit
Reliable Test Actions*



*Manage and Execute
Your Automated Tests*



*Reproduce Bugs and
Maintain Your Tests*



Why Use Ranorex

Watch Now at www.ranorex.com/why

Award-winning test automation tools, which allow testing of many different application types, including: Web browsers (IE, FF, Chrome and Safari), WPF, Flash/Flex, Silverlight, Qt, SAP, .NET, 3rd Party Controls and Java.