

PROFESSIONAL TESTER

SUBSCRIBE
It's FREE
for testers

Essential for software testers

| February 2011 | £ 4 / € 5 | v2.0 | number 7 |



Managing manual testing

Including articles by:

Bogdan Bereza-Jarociński VictO **Mohamed Patel** Equiem **George Wilson** Original Software
Derk-Jan de Groot Valori **David Yuill** HP **Erik van Veenendaal** **Ashwin Palaparthi** AppLabs
Marek Kucharski Parasoft

Award-Winning Test Automation Tools

“ Ranorex is the best *Commercial Functional Automated Test Tool* for .NET and Flash/Flex applications. ”
— 2010 ATI Automation Honors Awards




Object-based Capture & Replay Editor

- ✓ Maintainable recordings via the actions table editor
- ✓ Integration of Ranorex repositories



Automated Testing of Web & Windows Applications

- ✓ Winforms / C# / VB.NET
- ✓ WPF / Silverlight / Win32 / MFC
- ✓ Flash / Flex / Web 2.0 / AJAX /  / 



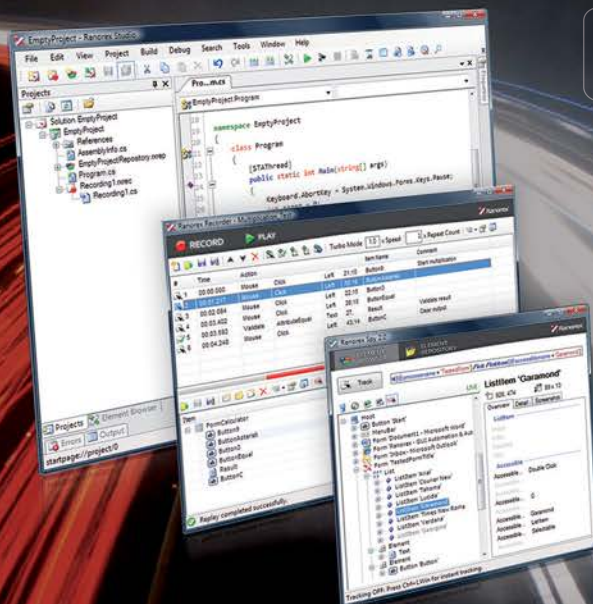
Maintainable UI Object Repositories

- ✓ Easy to maintain all types of UI objects
- ✓ Separate test automation from UI identification



Write tests in C#, VB.NET and IronPython

- ✓ Automatic and flexible code generation in C#, VB.NET and IronPython
- ✓ All-on-one test environment with code editor, code completion and debugging



Get your 30-day Trial
www.ranorex.com

Managing manual testing



Managing manual testing

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

Contributors to this issue:

Bogdan Bereza-Jarociński
Mohamed Patel
George Wilson
Derk-Jan de Groot
David Yuill
Erik van Veenendaal
Ashwin Palaparthi
Marek Kucharski

Publisher

Jerome H. Mol
publisher@professionaltester.com

Subscriptions

subscribe@professionaltester.com

Professional Tester is published
bimonthly by Test Publishing Ltd.

We aim to promote editorial independence
and free debate: views expressed by
contributors are not necessarily those of the
editor nor of the proprietors.

©Test Publishing Ltd 2010.

All rights reserved. No part of this
publication may be reproduced in any form
without prior written permission.

"Professional Tester" is a trademark of Test
Publishing Ltd.

The term manual testing, like automated testing, means more than just test execution. The idea that testing could ever be done without human intervention is dead: all non-trivial software models or at least interfaces with reality, yet can never match its complexity. That is why whenever people's interests are to be trusted to software, people will be needed to adjust testing to protect those interests better. This issue of Professional Tester is about making and controlling the adjustments.

As several of our contributors have noted in different ways, a key challenge of manual test execution is documenting it for repeatability and incident reporting. Using advanced

screen recorders such as *BBTestAssistant* (see <http://www.bbtestassistant.com>) is a fast-growing approach and we have five licences, worth \$199 each, for Blueberry Software's innovative and popular tool to give away. They will go to the first five readers to email me at editor@professionaltester.com identifying the story, book, TV programme or film to which each article headline in this issue refers.

Edward Bishop

Editor

SUBSCRIBE
It's FREE
for testers

IN THIS ISSUE

Managing manual testing

4 Robot, I

Bogdan Bereza-Jarociński proposes a new kind of tool

6 The edge of human

Hands-on performance testing with Mohamed Patel

10 We can remember it for you wholesale

Marek Kucharski on keeping manual execution traceable

13 Now wait for last year

Derk-Jan de Groot wants testers to be busier

15 Use the force better, Luke

David Yuill introduces HP *Sprinter*

19 Equality Unconquered

Power to the people, says George Wilson

Test process improvement

16 To maturity, and beyond

TMMi has recently been completed. Erik van Veenendaal tells us what is new

Test data

18 Better than life

Ashwin Palaparthi explains how AppLabs fabricates rather than prepares test data

Feature

22 Test library

Reviews of three new testing books and BSI's new web accessibility standard

 Visit professionaltester.com for the latest news and commentary

Robot, I

by Bogdan Bereza-Jarociński

What if the generation of tests for manual execution were automated?



Bogdan Bereza-Jarociński envisages an approach which reverses current common practice

Many articles in *Professional Tester* are concerned with improving automated dynamic test execution, because the potential benefits of doing that are well understood. However in practice a great deal of test execution is still done manually.

Some believe that will change, and more and more testing will become automated. They may be right: computers can be better than people at executing tests correctly, repeating tests consistently and checking results accurately, all of which are of course vital to effective retesting and regression testing.

Others think there must always be a place for manual test execution – for exactly the same reasons:

- sometimes inconsistent execution, inadvertent or not, increases coverage and therefore potential to detect defects which automated execution would miss
- sometimes people notice anomalies which a tool has been configured, wrongly, not to look for or to ignore because it was not foreseen
- automated execution can validate software, but a person can evaluate it: he/she adds business knowledge, understanding, intuition, imagination and empathy with users a tool cannot emulate
- the act of executing a test can lead a person to create additional valuable tests

- when time and resources are short, people can be asked to attempt the best testing possible under the prevailing conditions. In contrast tools usually have a fixed preparation and maintenance overhead which must be paid before they can be used to any advantage
- when part of a test cannot be run as written for an obvious reason, such as a minor change to interface design, a person can work around (when permitted, with care and raising an incident against the test) to complete other parts whose results may be the more important at the time. However trivial, such an obstacle usually stumps automated execution completely.

Some of these limitations of automation may diminish in the future: but few doubt that in the present at least some manual execution is essential.

So the variation in test execution and checking introduced by people is sometimes desirable, sometimes not. When it is not, how can we eliminate it? I suggest that the answer is by defining the tests more explicitly. Much of the weakness of manual execution comes from its association with manual test preparation. Even when standards and templates are used, test specifications have some room for interpretation. If that could be eliminated, manual testers could still use them as a basis for useful variations in both the actions taken and what they look for, but could be trusted far more to execute them correctly and not miss any significant discrepancy when needed. All the advantages and almost none of the disadvantages (the exceptions being speed and use of human resources) of manual testing would be realised.

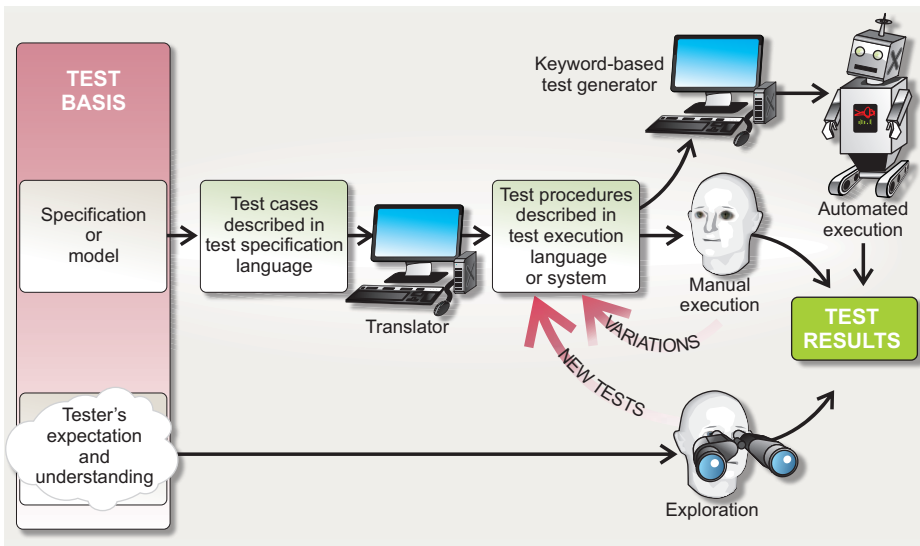


Figure 1: routes from test descriptions to executable tests – and back

Could this be achieved by using a tool to create consistent, unambiguous tests for people to execute?

Test specification – defining what the test is

To automate the creation of detailed test procedures, the test cases (pre-conditions, input, expected output and post-conditions) must be described unambiguously. Most formal languages used to define test cases are developed and used locally within an organization or even for a specific project. Some tool vendors provide basic frameworks for such languages, for example HP's *Business Process Testing*, which enables the creation of test cases as blocks of words which can then be manipulated graphically. They are not very much like programming languages, but more like business modelling languages, so that business rather than technical people can learn to write test cases and test scenarios using them.

Tailored languages can be made 100% suitable for the purposes of an organization and project. On the other hand, building, teaching and learning such languages is expensive, and they tend to hamper collaboration, so a standard language for this purpose would be desirable. Perhaps one could be based on a meta-language such as BPML or UML, or adapted from a language used to describe test cases such as TTCN or LabVIEW?

Test description – telling a person how to execute the test

Manual test preparation usually involves to some degree the use of natural language. This creates a lot of problems: different test analysts have different description styles; different organizations use different description guidelines; and different testers may understand a description differently.

Using a formal language at this stage too should eliminate the first two problems but would only change the third. The formal languages used to specify tests are designed for machine, not human, readability. Executing tests expressed in them manually would be difficult, painstaking, error-prone work. The keyword-driven approach, designed to make it easier to create and maintain automated tests, may be a partial solution, but the person executing would need to either know, or make constant reference to, the definitions of the keywords. Again this would probably be excessively demanding work in most circumstances.

So a second language is needed: more abstract, easily readable but still formal enough to define detailed procedures with no ambiguity. It may resemble natural language, express the actions and inputs

in some very ergonomic tabular or graphical form, include visual cues, and/or communicate with the person executing some other way. There is an opportunity here for great design but it must not go too far: for example, having a person repeat inputs shown in a sequence of images, or even a movie or similar, might take too much of his or her attention away from the test item.

Given the definition of such a language or other description system – which because of its purpose would need to be both simple and small – it should be quite easy to write a “translator” program that generates it from the test specification language.

Ideally, it would be possible to define new tests directly using the “execution language” or system too. It would be extremely useful in incident reporting and retesting when an execution-time variation of the procedure, or a new test created in an ad-hoc or exploratory way, detects an anomaly, or when it is desired to add such a test, passed or not, to a regression suite. Depending on the form the language or system takes, an extra interface or “development kit” might be necessary to achieve this, and/or syntax checking tools could be used to verify and debug “handcoded” tests.

Finally, the execution procedure could be used also as input for automated generation of tests for automated execution, on the same principle as keyword-driven automation methods. Thus the same tests could be run manually or automatically as most appropriate for the current objectives. Doing both and comparing the results, such as resulting change to back-end data etc, could be interesting too: it may help to reveal some subtle and dangerous defects such as timing issues that either manual or automated execution alone cannot

Bogdan Bereza-Jarociński is a testing consultant, speaker and trainer and a long-time contributor to Professional Tester. He is the proprietor of VictO (<http://victo.eu>)

The edge of human

by Mohamed Patel

Applications are mutating in unpredictable ways. Performance testing must be able to adapt



Mohamed Patel tells us about his favourite tool

After many years as a performance tester I have learned to expect the unexpected. While most other testing disciplines aim for repeatability and predictability, performance testing has always been about ad-hoc problem solving. We operate not on the clean, well-lit superstructure of user interfaces and architecture designs but in the very murky depths. These days even most developers don't know what goes on at low level as they assemble their applications from dinky components and flashy development kits.

Constructing effective test scripts from protocol transactions requires peering into dark recesses, and whatever bizarre things are found must be simulated by many replicants, using

complex logic and data handling, each behaving realistically but differently: simple cloning usually won't do. While functional testing may be (very slowly) moving towards standardization, performance testing is diversifying. It's been a long time since I have worked on two similar projects. Rather, it's amazing how different each new situation – ie application and testing requirement – is from all the others I've seen before.

The power to change

So increasingly, assuring performance requires not systematic skills or prescriptive tools, but extreme flexibility. In order to deliver the testing required, the tester must be able to adapt and innovate methods and to override and extend automated functionality. At Equiem we use and consult on many performance testing tools and are often asked which we prefer. On a simple comparison of features, there is often little to choose between them: some are slightly stronger than others in various areas, but not importantly so. A good fit with development and other testing technologies in use can be a factor too. But to us the vital thing is *extensibility*: the capacity to create the behaviours you need, rather than paying for many built-in ones you don't. On that criterion, the leading tool is Facilita's *Forecast*.

Test implementation using actual application code

Using Forecast's capabilities to the full requires coding. We don't see disadvantage in this as we believe it is a skill the modern performance tester simply must possess. For example, DLLs, JARs or .NET assemblies can be associated with custom virtual users. The external code then becomes available for use within the scripts: the developer's, or third-

Is your test process ready to cope with increased workload?

According to our recently published World Quality Report 2010-2011, in co-production with Sogeti and HP, investment is shifting towards building new applications¹, which means process improvements are necessary in order to cope with the increasing workload.

One way to achieve the necessary improvements is using the Test Process Improvement model - TPI NEXT[®]. Developed by Sogeti, TPI NEXT[®] is our world-leading model for providing an objective step-by-step guide to business-driven test process improvement.

How the TPI NEXT[®] model works

The TPI NEXT[®] assessment is used to measure your testing process. How mature is your organization at a particular moment? Which business drivers need to be addressed? Interviews and accelerators assist in creating a target maturity matrix. This provides an overview of the Key Areas that should be improved in order to reach a higher maturity. These are prioritized and the corresponding improvements and implementation support is defined. This approach has already been successfully applied at large international clients such as Air France-KLM. General conclusions drawn from these assessments supports implementation of the model in future assessments.

Conclusions drawn from carrying out TPI[®] NEXT scans

- The TPI NEXT[®] model is highly suitable for tailor-made scans for organizations and businesses.
- Ensure that the people being interviewed know up-front that this is not an audit – people themselves are not being judged!

- TPI NEXT[®] scans require thorough planning, especially when there is a short timeframe and stakeholders reside in different countries.
- Implementing the improvements after the TPI NEXT[®] assessment needs attention and commitment from management.

Beyond TPI NEXT[®]

Capgemini has extensive experience in providing a clear visualization of the improvements and implementation that result from a TPI NEXT[®] assessment. By implementing the conclusions from TPI NEXT[®] scans and clarifying the roadmap to our clients, it becomes easier to evaluate and check the necessary improvements. This approach combined with the full commitment of the Capgemini team has proven to be especially appreciated by our clients.

Spreading the experience

At Capgemini in the Netherlands, the TPI Expert Group is currently developing courses to help clients put our experience into their practice. Researching and combining different test process improvement models and best practices together with Capgemini's Quality Blueprint (which provides a comparative benchmark against the industry standard) lead to practical support and guidance throughout the improvement process.

For more information about TPI NEXT[®] and the activities of the Capgemini TPI Expert Group, please contact us at testen.nl@capgemini.com.

¹ http://www.uk.capgemini.com/insights-and-resources/by-publication/world_quality_report_2010__2011



party, libraries can be used to encode and decode data during testing without needing access to proprietary or otherwise unavailable source code.

Object-oriented scripting

The class structure of the scripts enables the tester to override any of the base methods, introducing his or her own logic, conditions and validation. I used this recently when the application under test required client-side timestamping and pre-validation of data before every HTTP POST request. Coding this once, in the custom virtual user, means it is automatically implemented in all requests sent by all scripts, including ones yet to be created. These concepts are of course nothing new to OO programmers, but many other performance testing tools try to hide the real code behind user interfaces or simplified procedural languages that serve only to restrict what you can do.

Global editing

Heavily UI-based tools can be very cumbersome, requiring user input for every data item to be correlated or modified, making for a great deal of error-prone editing. Instead, Forecast has a wizard to define script generation rules. When a pattern, eg a header type, URL or specific document content is detected, the rule inserts code for correlation, checking, extraction, header creation and so on. Rather than editing the scripts, one edits and extends the rules: the scripts are then regenerated according to the new rules.

Dynamic form handling

Imagine a large form with many fields, perhaps containing details of a retrieved customer account, and a test that requires one field to be amended before the form is submitted. In other tools, the script contains code to populate all fields. A Forecast script refers to the one changed field only, making it easier to edit, extend or re-use. The other field inputs are correlated automatically with the values embedded when the form was served. They are in the script only as comments: if desired, this behaviour can be changed,

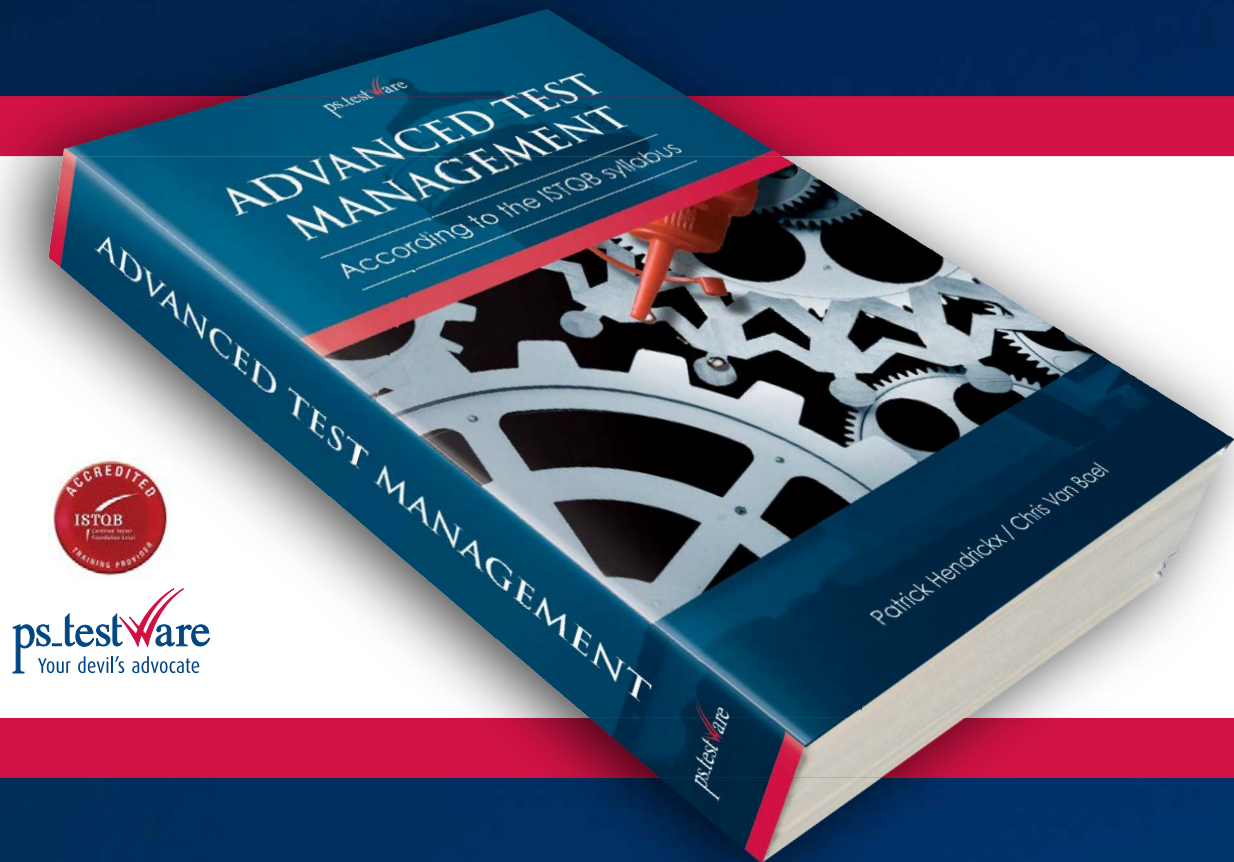
causing the script always to amend more fields.

Now, suppose the form is designed to change depending on the data: that is, it can have more, fewer or different fields depending on the customer type and/or history. With many tools, it is necessary to determine every possible variant of the form – which itself can be difficult or impossible – and create specific scripts to handle each. With Forecast, provided the fields they amend are still present, scripts always execute. Values of all other fields, no matter what they are, are captured when the form is served and sent automatically when it is submitted ■



Mo Patel's 25-year IT career has included successful performance testing of many complex applications in the retail, banking and public sectors. He is a founder and director of Equiem (<http://equiem.com>) which specializes in highly tailored performance testing services. For more information about Forecast see <http://facilita.co.uk>

For testers and test managers who want to enhance their knowledge of test management



In their strivings for operational efficiency, quality and to satisfy growing government regulation, the number of companies that test software professionally is growing.

In "Advanced Test Management" testers and test managers will find :

- an overview of various approaches and techniques
- numerous examples, tips and tricks, tables and illustrations

The book provides a clearer and more effective manual for a well-oiled testing approach. This knowledge allows you to arrange custom software testing and integrate it in any business environment.

The book ties in with the knowledge needed to gain the ISTQB Advanced Test Management Certificate in Software Testing. ps_testware is an accredited ISTQB Foundation and Advanced training provider.

HOW TO GET IT?

You can order this book for **44,95 EUR*** via www.pstestware.com or get it for free when attending our **ISTQB Advanced Test Management training**.

* (excl. VAT and shipping costs)

ps_testware
Your devil's advocate



ps_testware is a leading company specialized in software testing, software quality and quality assurance. With offices in **Belgium, The Netherlands and France**, ps_testware provides services in all matters of structured software testing and related fields.

For a detailed table of contents: www.pstestware.com

We can remember it for you wholesale

by Marek Kucharski

To stay on track, trace



Many were surprised last autumn when Parasoft embraced manual testing. **Marek Kucharski** explains what has changed

There is a Polish saying that “only a cow does not change her mind”. She is happy just to chew grass. Parasoft, a company known for advanced tools, has been like that cow for years: we thought that dedicated testers would eventually be replaced almost completely by tests

created by developers and executed automatically overnight. We've changed our mind. We still believe in automation, and we use our products to automate a very high proportion of our internal testing. But we now acknowledge that sometimes manual testing is the best, or even the only, option and have extended our ALM platform, Concerto, to embrace it: making it as traceable, auditable and integrated with development as automated testing.

Visibility makes faster work

Figure 1 shows a requirement in Concerto. The tabs at the top summarize and give access to detailed information about the work done so far. To implement the requirement, 32 development tasks were identified (these include all development work, not just coding); 37,641 lines of code have been created or modified; two automated tests have been run and detected no defects; and nine manual tests (shown under the Scenarios tab) have been run, of which four have failed.

Thus the people executing the manual tests have visibility of what everyone else, including developers, has already done:

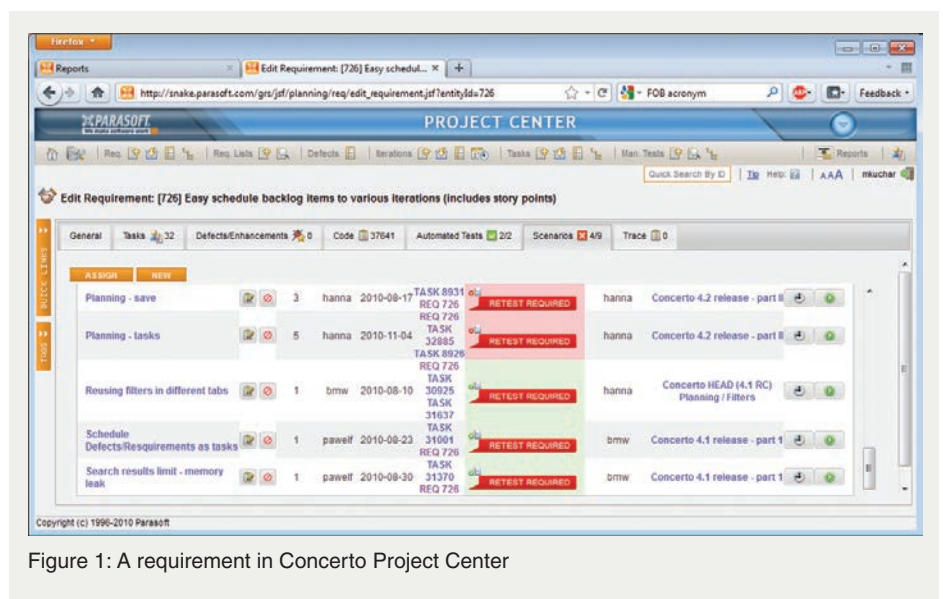


Figure 1: A requirement in Concerto Project Center

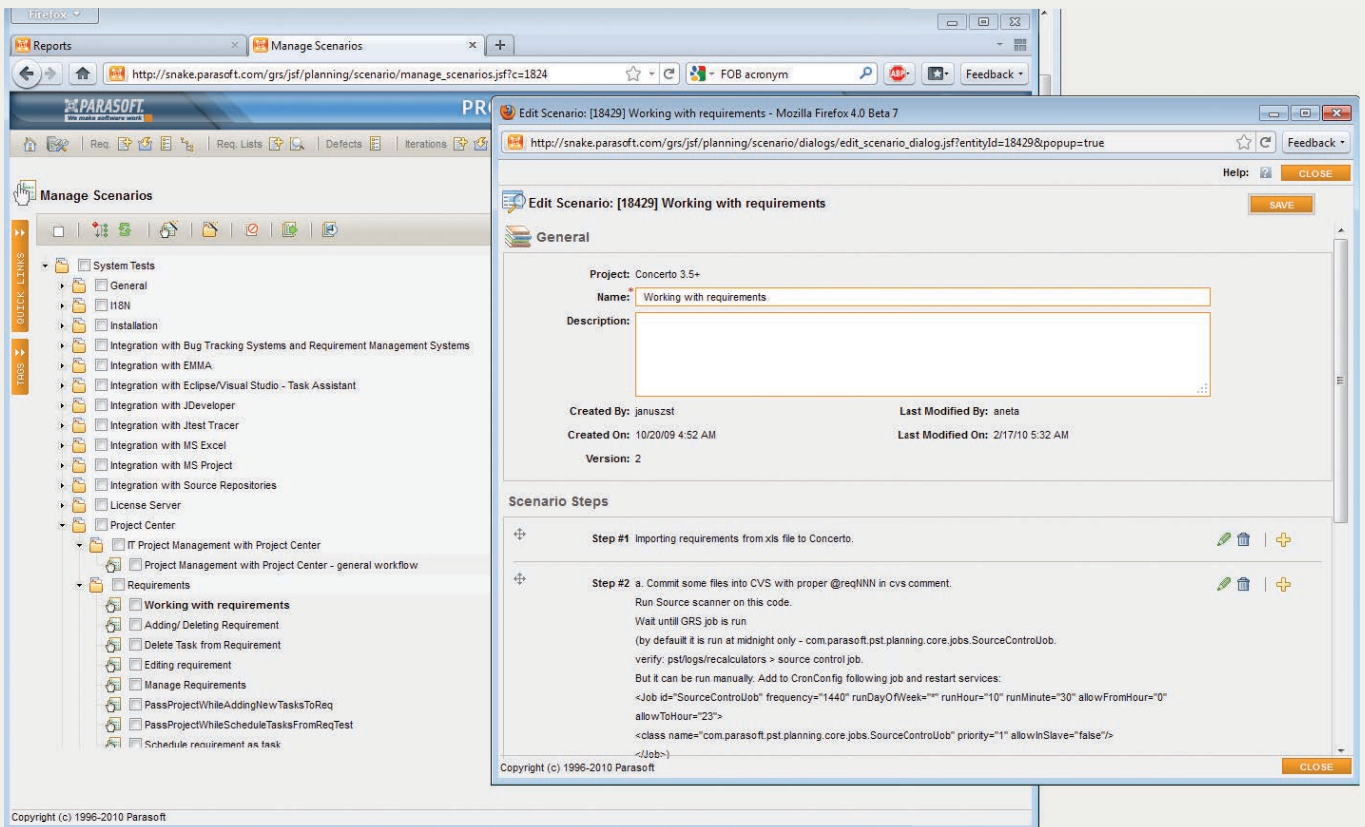


Figure 2: manual test scenarios being managed and edited

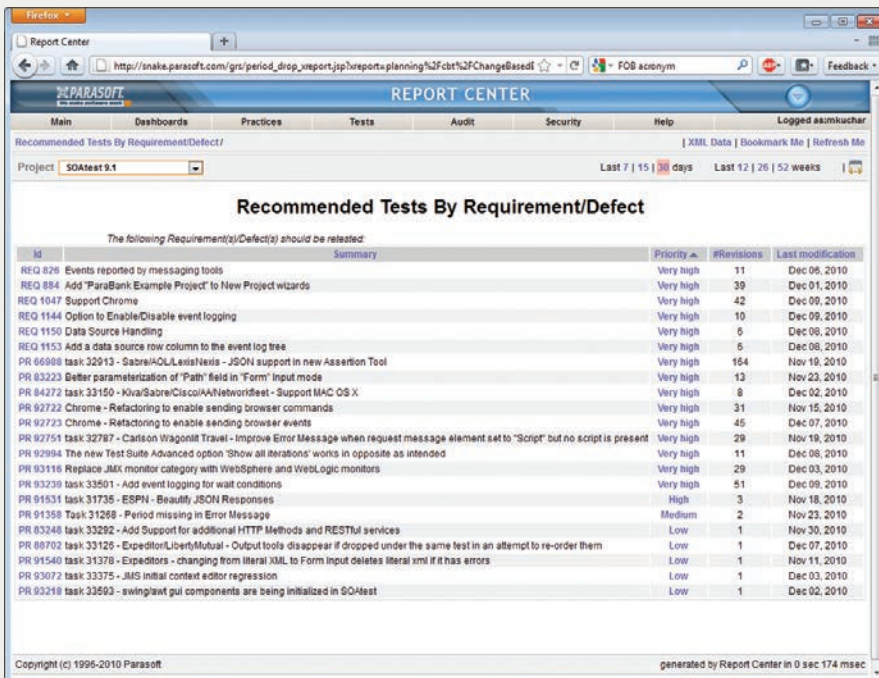


Figure 3: retests recommended due to code modification

manual or automated. The steps taken and their outcomes are recorded: they demonstrate to the tester what is considered correct behaviour, far more quickly and clearly than a formal description. This avoids misunderstandings and helps the tester know what is and is not an incident. The tester adds additional scenarios, based or not on the ones provided by development.

Traceability makes less work

Then, the loop is closed: when a defect is detected and fixed, Concerto enables traceability of test to requirement, requirement to code, and defect to modification. It therefore knows at all times exactly what tests, both manual and automated, need to be re-executed for retesting and regression testing purposes (figure 3). This information enables enormous savings in what is by nature a time-consuming activity. Finally, a static analysis rule can be created to prevent the construct(s) that caused it being repeated anywhere else. Collaboration at that level between development and QA makes future expensive defects simply not happen.

unit testing, static analysis, regression coverage and everything else. This input to manual testing helps to target it precisely, making it easier and more cost-effective. When a developer completes a task, he or she creates a test for it,

Managing manual testing

Tracer makes traceability work

These facilities are available even when code is created by an external development organization which is not using Concerto. *Tracer* identifies the methods and objects used when each test case is executed, mapping test cases to code, completing the traceability information needed to manage and report (figure 4) the entire development and test effort, including manual testing, comprehensively ■

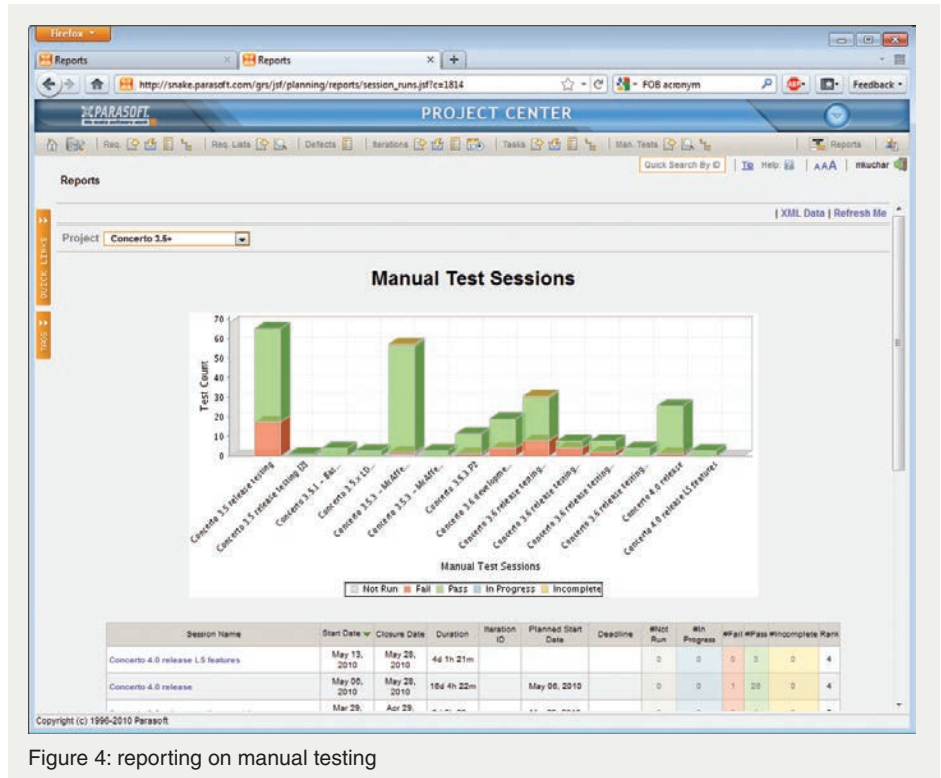


Figure 4: reporting on manual testing

Marek Kucharski is CEO Europe of Parasoftware. For more information about Concerto see <http://parasoftware.com/concerto>

READY

10 PRINT "SUBSCRIBE TO PT"

20 GOTO 10

RUN ■

PROFESSIONAL
TESTER

Essential for software testers

Now wait for last year

by Derk-Jan de Grood

The things that delay testing and how to avoid them



Why are we waiting?
Derk-Jan de Grood
finds out

Software testing is usually on the critical path. Most testers feel the burden of the anxiety and impatience of managers and stakeholders, who expect testing activity to reflect it, and often express surprise to find this is not the case: when the storm everywhere else is reaching its height, the testers... *wait*. Why, when even small delays can have severe impact on the project timeline? Because while test execution is given high priority, providing the things needed to do it is not. Not having those things at the right times (i) delays the start of test execution; (ii) makes test execution take longer; and (iii) makes testing less effective and dependable by requiring more assumptions to be made.

Good test managers try to emphasize the advantages of early involvement and working in parallel with development, so that test execution can begin immediately and be done efficiently whenever work products are released. Unfortunately those concepts are still not well understood – or are not taken seriously – by other leaders, whose concern with project timescales tends to make them concentrate on removing any potential cause of delay to development, forcing more of the testing work to take place later, actually causing worse delays.

To help managers to reduce time-to-market, we as testers need to get the measures required for better, more timely testing higher on their agenda. Trying to do this by making them understand testing better has failed. We might achieve more by focusing instead on what they do understand, highlighting the project issues

that cause delay to testing and what might be done to eliminate it.

I recently carried out questionnaire-based research with testers in multiple industries, aiming to discover the causes of the wasted waiting time. This article discusses the three cited most often, and suggests approaches to arguing for project change that might help to mitigate them. That should keep testers busy for more and unduly pressurized for less, of the time, helping to bring about what everyone wants – quality products delivered faster.

Unavailability of test environment

Test execution time is typically greatly increased because the test environment is unavailable, unstable or unusable.

This is a familiar but still common situation for testers. I recently worked on a large project with a one-week release cycle. Unfortunately “release” meant only delivery of code. The deployment and configuration required to enable meaningful test execution and results checking took several days, reducing testing time to two or three days a week.

Unstable and slow environments are also commonly experienced. Both cause very significant delay and risk. In the first case, the environment crashes before the test is complete so it must be repeated unnecessarily. Casual testers – ie stakeholders and developers – who “explore” software often do not understand that “carrying on where you left off” is usually impossible and always dangerous in systematic testing. As well as wasting their time waiting for responses, sluggish interfaces affect testers' concentration and lead to mistakes.

Discussions with project managers regarding test environments tend to be rare

because they are seen as a side issue, not contributing directly to the delivered product. It needs to be made clear that if release and testing of code are on the critical path then so is making the release testable. The concept of a timeline event called "release to testing" – which occurs only after (i) the developers have released not a build but a full installation and (ii) the testers have verified it might help. Failing that, asking the following questions will help to anticipate problems and taking action to make more of the answers positive will reduce execution time.

- are test environment considerations being included in project risk management?
- has the number of environments, and instances of each, been established or estimated?
- does the test plan include identification and the project plan creation and maintenance of test data?
- have the specifications of environments yet to be created been documented and agreed?
- have the stability and performance of environments already created been assessed?
- will there be any requirement to share environments with other activities or projects?
- does the project plan include allocation of environments to testing and does the schedule show the associated dependencies?
- have technical support resources required to provide and configure user accounts and then to assist users of the environments?
- has configuration management to enable environments to be reproduced exactly, and to track change and difference between environments, been implemented?

Delay in fixing show-stopping bugs

Testing aims to find the most important bugs first, but those bugs are often showstoppers which cause testing to be suspended until they are fixed. But, how long will that take? It's hard to say. Bug fixing is seldom a well-managed process. It's important to ensure that management realises and takes into account the fact that while incidents are being reported, reproduced, discussed and resolved, testers will often be waiting. When discussing this problem with management, ask the following questions:

- is the incident management mechanism sufficient and are all who should be using it correctly?
- if an incident occurs and a tester is unsure whether to raise it, what should he or she do?
- does the project plan allocate sufficient time for incidents to be resolved (investigated and, if necessary, fixed)? Do its estimates take into account quality, complexity and commenting of code and availability of the people needed to resolve incidents?

Lack of sufficient information about the system

Every tester understands the importance of getting system definitions (specifications, requirements, use cases, stories etc) as early as possible. If they are late, test preparation is made difficult, causing subsequent time-consuming change. If they must be chased, the time for test preparation is reduced, with the same result. If they are inadequate, the time taken to discuss incidents is increased, impacting bug-fix time.

To create and justify a strategy that will not fail because of waiting caused by unsafe assumptions, ask management:

- expressed quantitatively (eg on a scale of 1 to 10, where 1 is a blank sheet of paper and 10 is the complete, perfect design), how detailed can we expect documented requirements to be (i) before the testing project begins; (ii) at management-defined milestones in the critical path of the development project?
- how accurate can we expect documented requirements to be, as a proportion of the final features of the accepted product, at those critical milestones?
- is the information needed by the various project participants, including testers, identified and agreed before system definitions are documented?
- where detailed documentation of that information is not to be available, are information sharing activities to replace it planned?

Other reasons testing time is lost

The research identified four other common issues that force testers to wait. The full results and many more suggestions are included in a comprehensive checklist, intended to offer a fresh approach to opening and maintaining productive dialogue. Its questions align with the concerns and areas of expertise of project management, helping to eliminate time wasting from any testing effort. The checklist is available free at <http://www.smartest.nl/toolstemplates/procesverbetering> ■

Derk-Jan de Grood is a test manager at Valori (<http://valori.nl>) and author of TestGoal: Result-Driven Testing (Springer, ISBN 9783540788287). His new book in Dutch, Grip op IT: De Held Die Voor Mijn Nachtrust Zorgt (Academic Service, ISBN 9789012582599) will be published later this year. He speaks frequently at international testing conferences, including about his passion for aligning IT and business

Use the force better, Luke

by David Yuill

Manual test execution is monotonous, time consuming and error prone. Why is it still so common?



David Yuill introduces HP's new concept: accelerated manual test execution

In certain situations manual testing is better than automated. It can take many forms, adapting to achieve immediate objectives and solve or work around problems at any point in the application lifecycle, making it popular with agile development teams and with V-model-minded testers. Some of those forms require little preparation and none require script recording, coding or intricate technical setup. Some do not require technical skill: there will always be parts of applications that must be tested manually by business analysts and end users as well as testers. There will always be the need to check for important defects very quickly using knowledge and experience rather than systematic techniques. And testers will always be expected to and want to explore products in creative, unplanned ways to improve assurance against unforeseen high-impact failures. Entirely manual testing also has disadvantages, but they are greatly reduced by HP's new *Sprinter* technology, which is now core functionality in Quality Center.

Data-driven manual testing is inaccurate

A wrongly-performed step or incorrect data input can lead to overlooked defects or wasteful false incidents. Repeating the same steps multiple times with different inputs makes mistakes even more likely, because of the sheer tedium and the need to switch attention between the application under test and the data source. As time becomes short discipline is lost under pressure to take shortcuts, deliberately skipping steps or entering incorrect data. HP Sprinter, under manual control,



automatically injects the correct data into every field, increasing speed, accuracy and ease.

Manual compatibility testing takes too long

It is typically possible to execute manual tests on a very limited number of environments: there simply is not time to continue to repeat execution. HP Sprinter's mirror testing replicates manual execution automatically and simultaneously across multiple platforms and configurations, increasing compatibility coverage.

Reporting and reproducing incidents wastes time

Whenever manual execution has a nonsystematic element, reporting an incident sufficiently becomes difficult. That delays resolution which in turn delays testing. Sprinter records and logs manual testing steps precisely ensuring that every incident reported is reproduced at the first attempt. The recording is easy to read and HP Sprinter provides state-of-art screen and movie capture and annotation facilities to accelerate test documentation and incident management and resolution ■

David Yuill is Apps Product Solution Marketing Manager (EMEA) at HP. For more information about HP Sprinter see <http://hp.com/go/sprinter> and <http://youtube.com/watch?v=-G8C61PnIS0>

To maturity, and beyond

by Erik van Veenendaal

TMMi intends to help organizations achieve more effective, more efficient, continually improving testing. The first complete version was launched last month



Erik van Veenendaal describes the final two maturity levels which have been added

The Test Maturity Model Integration (“TMMi”) is a guideline and reference framework for test process improvement. Such a framework is often called a “model”, that is a generalized description of how an activity, in this case testing, should be done. TMMi can be used to complement Capability Maturity Model Integration (“CMMI”), the Carnegie Mellon Software Engineering Institute’s wider process improvement approach (see <http://sei.cmu.edu/cmmi>), or independently.

Applying TMMi to evaluate and improve an organization’s test process should increase test productivity and therefore product quality. In achieving this it benefits testers by promoting education, sufficient resourcing and tight integration of testing with development.

Like CMMI, TMMi defines maturity levels, process areas, improvement goals and practices. An organization that has not implemented TMMi is assumed to be at maturity level 1. Being at level 2, called “Managed”, requires the practices most testers would consider basic and essential to any test project: decision on approach, production of plans and application of techniques. I call it “the *project-oriented* level”.

The goals and practices required by level 3, “Defined”, invoke a test organization, professional testers (that is people whose main role is testing and who are trained to perform it) earlier and more strategic test planning, non-functional testing and

reviews. These practices are deployed across the organization, not just at the project level. I think of level 3 as the one where testing has become *institutionalized*: that is defined, managed and organized. To achieve that, testers are involved in development projects at or near their commencement.

Version 3.1 of TMMi, launched at EuroSTAR in December 2010, defines its top levels: 4 “Measured” and 5 “Optimization”.

TMMi level 4: Measured

This is the level where testing becomes *self-aware*. The *Test Measurement* process area requires that the technical, managerial and operational resources achieved to reach level 3 are used to put in place an organization-wide programme capable of measuring the effectiveness and productivity of testing to assess productivity and monitor improvement. Analysis of the measurements taken is used to support (i) taking of decisions based on fact and (ii) prediction of future test performance and cost.

Rather than being simply necessary to detect defects, testing at this level is *evaluation*: everything that is done to check the quality of all work products, throughout the software lifecycle. That quality is understood quantitatively, supporting the achievement of specified quality needs, attributes and metrics. Work products are evaluated against these quantitative criteria and management is informed and driven by that evaluation throughout the lifecycle. All of these practices are covered in the *Product Quality Evaluation* process area.

The *Advanced Peer Reviews* process area is introduced and builds on the review practices from level 3. Peer review

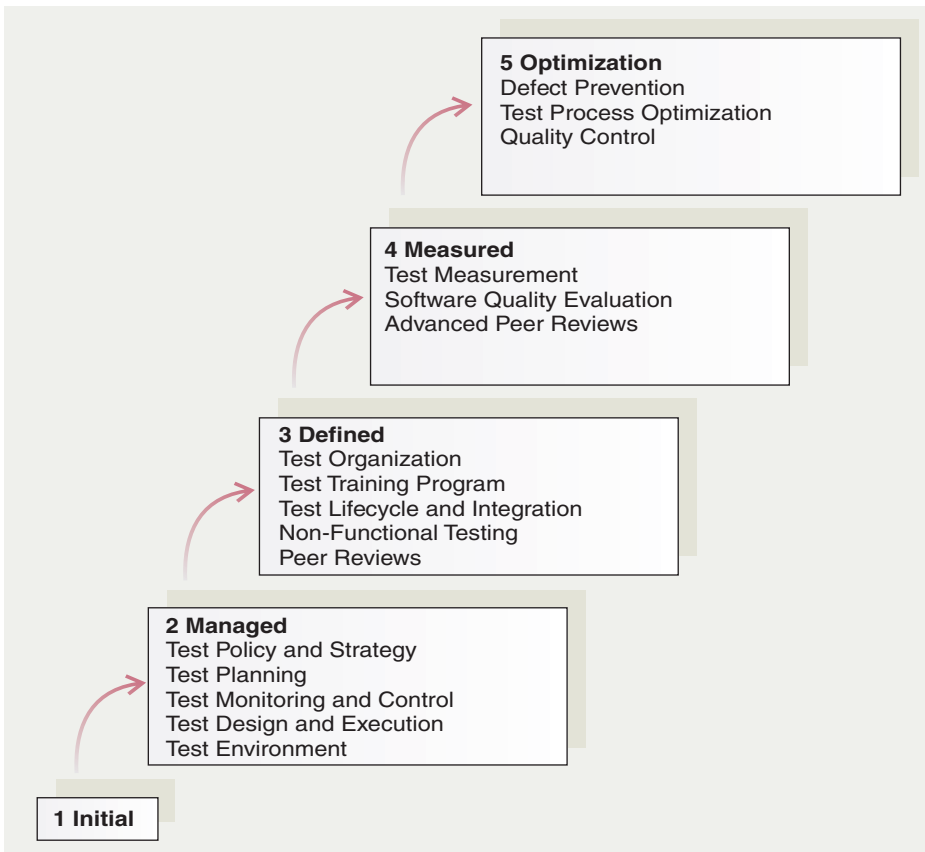


Figure 1: TMMi maturity levels and their process areas

becomes a practice to measure work product quality early in the life cycle. The findings and measurement results are the basis of the strategy, planning and implementation of dynamic testing of subsequent (work) products.

TMMi Level 5: Optimization

When the improvement goals at levels 2, 3 and 4 have been achieved, testing is defined completely and measured accurately, enabling its cost and effectiveness to be *controlled*. At level 5 the measurements become statistical and the control detailed enough to be used to fine-tune the process and achieve continuous further improvement: testing becomes self-optimizing.

Improvement is defined as that which helps to achieve the organization's business objectives. The basis for improvement is a quantitative understanding of the causes of variation inherent to the process; incremental and innovative change is applied to address those causes, increasing predictability. An optimizing process is also supported as

much as possible by automation and able to support technology transfer and test process component reuse.

To achieve such a process a permanent group, formed of appropriately skilled and trained people, is formally established. Some

organizations call this the *Test Process Group* or TPG: it relates to and grows from the test organization defined at TMMi level 3, but now takes on responsibility for practices introduced at level 5: establishing and applying a procedure to identify process enhancements, developing and maintaining a library of reusable process assets, and evaluating and selecting new test methods and tools.

Level 5 introduces a new process area, *Defect Prevention*. Defects are analyzed to identify their causes and action taken, comprising change to the test and/or other processes as necessary, to prevent the introduction of similar and related defects in future. By including these practices, at level 5 the objective of testing becomes to prevent defects.

This and the other process areas introduced at level 5, *Test Process Optimization* and *Quality Control*, are interdependent and cyclic: Defect Prevention assists product and process Quality Control, which contributes to Test Process Optimization, which in turn feeds into Defect Prevention and Quality Control. All three process areas are, in turn, supported by the continuing practices within the process areas established at the lower levels ■

Erik van Veenendaal (<http://erikvanveenendaal.nl>) is a widely-recognized expert in software testing, an international testing consultant and trainer and the founder of Improve Quality Services BV (<http://improveqs.nl>). He is the lead author and developer of TMMi and vice chair of the TMMi Foundation. His new book with Jan Jaap Cannegieter, The Little TMMi: Objective-Driven Test Process Improvement, is reviewed on page 22

Better than life

by Ashwin Palaparathi

Real data limits testing: fabricated data empowers it



Ashwin Palaparathi explains how AppLabs creates the data it needs to test enterprise-level applications

Getting test data with enough volume, variety and variability is often troublesome, more so when testing multi-environment enterprise systems that will interface with external systems. Using real data has compliance implications and adapting it to deal with them properly often compromises testing effectiveness. The painstaking work done to make the data “safe” and extend it for instrumentation purposes while maintaining integrity and dependency is very expensive to repeat when change to the application under test occurs.

To address this challenge AppLabs creates test data from scratch, using its own Data Fabrication Toolkit (“DFT”) to produce the very large numbers of records commonly necessary for testing in banking, financial, insurance and healthcare applications, or to test database performance or validate analytics in any system. DFT is integral to our service delivery. It includes features to populate and maintain referential integrity of specific, difficult fields including US Social Security number, UK National Insurance number and credit card details. The data can also be very rich. DFT includes facilities to calculate and insert values that:

- are functionally representative, in order to achieve coverage of classes and domains
- violate defined constraints, for robustness and reliability testing
- contain security threats such as SQL injection and persistent cross-site scripting attempts.

Sculpting and controlling the data

The inputs to DFT include XML files containing the field definitions plus captured metadata that controls the quality, variety, and variability factors such as referential integrity, geographical and demographic variation and business intelligence. Its configuration controls include support for static configurable lookup, and weighted-random pickup of data from enumeration sets.

As well as populating test databases, DFT can create related test input data (figure 1). In recent client projects we have used this capability to create data-driven test suites – for both manual and automated execution – to invoke and exercise specific combinations of input and test data, and meta-driven test suites

to permute the order in which test cases are executed with each regression cycle. Test inputs can be varied using fully configurable randomization too.

Deploying, refreshing and updating

Loading data produced by DFT into the test DBMSs is automated using Apache ANT (<http://ant.apache.org>)

Once DFT has been configured to produce the required data, the same configuration can be used again but with the addition of uniformly-distributed or stochastic randomization. This creates further data which has the same defined characteristics and is governed by the same constraints, but is materially different, refreshing the test data so increasing the coverage and defect finding potential of testing.

To update data in a managed rather than random way, the characteristics that make each record valid or invalid are recorded and can be varied at will: so the minimum amount of change to make valid records invalid and vice-versa can be applied easily, and coverage of the range of factors that make them so monitored. A second approach uses a small amount of seed data to ensure the presence of specific, desired records among the many created on-the-fly.

The test data is under full configuration management at all times: DFT is integrated tightly with CVS (<http://nongnu.org/cvs>) ■

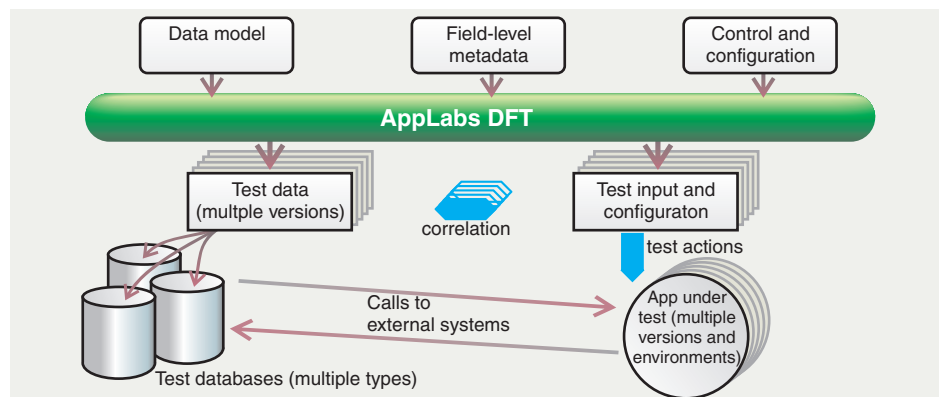


Figure 1: DFT in test and test data generation

Ashwin Palaparathi (ashwin.p@applabs.com) is VP, innovation at AppLabs, which he rejoined recently when it acquired ValueMinds, the company he left to found three years ago and which has created many innovative test tools including testersdesk.com

Equality Unconquered

by George Wilson

Testing will never stop diversifying and never should



George Wilson says quality management tools should provide liberation, not limitation

Application Quality Management (AQM) products have many functions. They do different things for people in different roles, depending on complex factors such as process, entities and other tools. However their purpose is clear and unchanging: *to ensure business objectives are met*. Doing that well is becoming harder as software organizations are increasingly challenged to achieve better quality, less risk, faster delivery and lower costs.

So development methodologies are evolving, becoming more agile and closely aligned to changing business need. Testing is having to change too but, as always, how it should is less obvious. As development becomes more reactive and unpredictable, the ways in which testing is organized and performed across organizations, teams and even projects are becoming more, not less, varied and complex. To cope, testing must remain able to change itself, to integrate more and more closely with project

management, development and operations and to involve business more and more directly. The days when these functions operated in their own silos and communicated infrequently are gone. Intrinsic continuous connection between them is now mandatory.

AQM has not kept up. The market-leading products continue to impose their own processes: a narrow and limiting hierarchical workspace of requirements, tasks and defects based on the practices of the programmers that created them or on dubious interpretations of incomplete, decades-old standards. At Original Software we formed the opinion some time ago that what testers need now is not a prescriptive database application, but a quality management platform that can be used to implement and support any process and integrate with any external activity or tool. Our offering, *Qualify*, was launched last year. Like older products it stores, monitors, controls and communicates information about requirements, design, build, test planning and control, test execution, test environment and deployment, providing a unified view. But it is designed for use by business analysts, project managers and operations as well as development and testing staff, enabling all to implement their own processes exactly and assimilate them seamlessly.

Choose your own adventure

Qualify's data definitions are completely configurable: its flexibility is limitless. It comes with templates based on all the popular methodologies, including traditional ones, for customization, or can be set up from scratch within realistic time: a customer with expertise in Sogeti's TMap implemented it fully using Qualify in 48 hours. A user account, once created, is available at all times, even across different

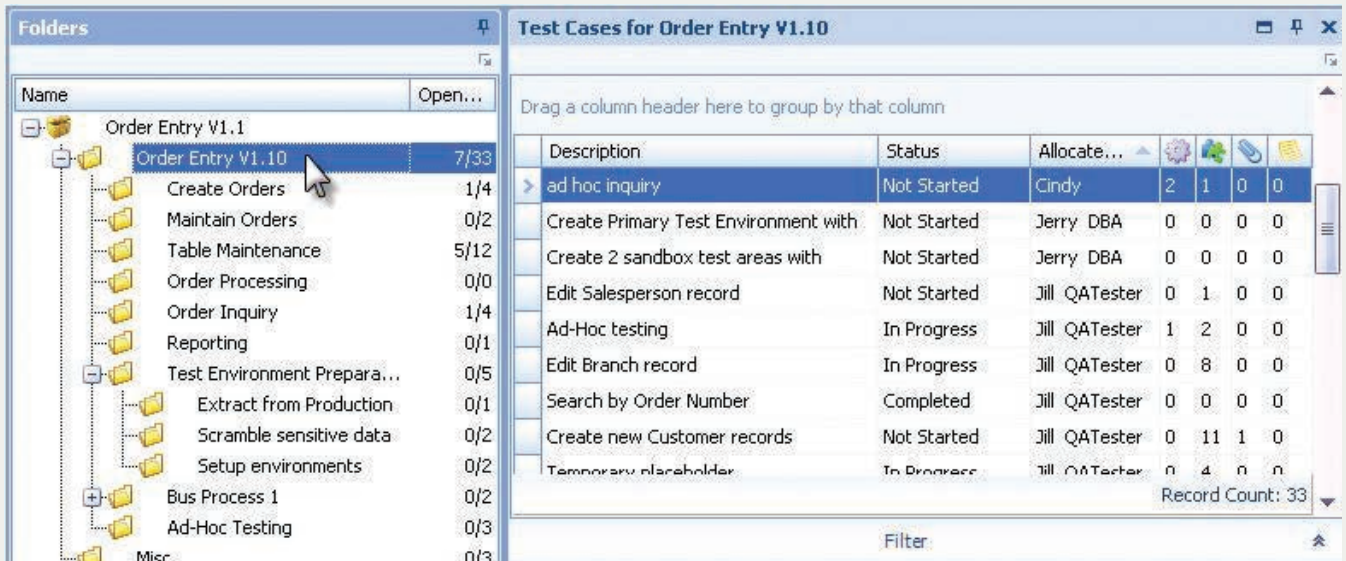


Figure 1: test cases for requirement "Order Entry 1.10"



Figure 2: grouping test cases

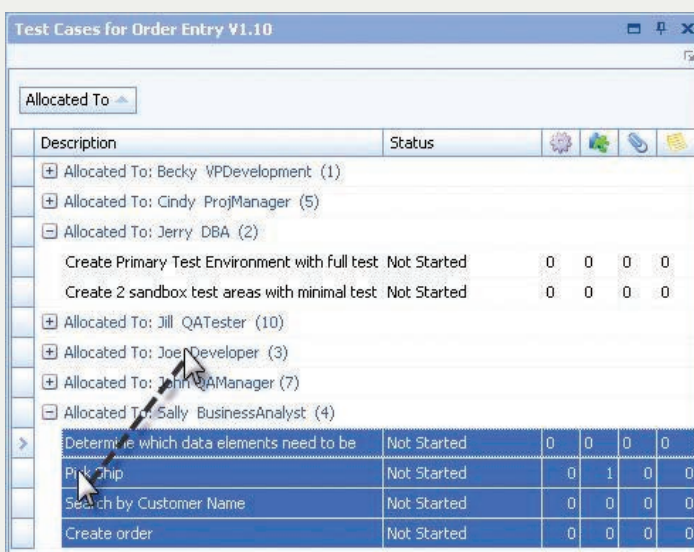


Figure 3: reassigning multiple test cases

methodologies and roles. Its attributes are retained and its permissions can be configured separately for each project. While other tools require a great deal of data input before testing effort can begin, and continue to provide more questions than answers for a long time after that, Qualify hits the ground running. This flexibility is particularly valuable to testing consultancies and service providers: they can provide the test process each of their customers prefers using a single product and re-using assets and expertise common to multiple projects. And when an improvement to a process is identified, it can be implemented immediately.

To each according to need, not ability Direct, objectives-driven management requires removal of unnecessary barriers between roles. Rules such as that only a test manager can assign test cases to testers, only developers can change the status of an incident to "fixed" and only a tester to "retested", and testers produce summary reports for BAs to read are simply not agile. The idea that anyone in the team can take on any of the team's tasks, is. Qualify's extreme ease of use makes that a reality: its interface is clean, simple, intuitive and completely code free. Here's an example:

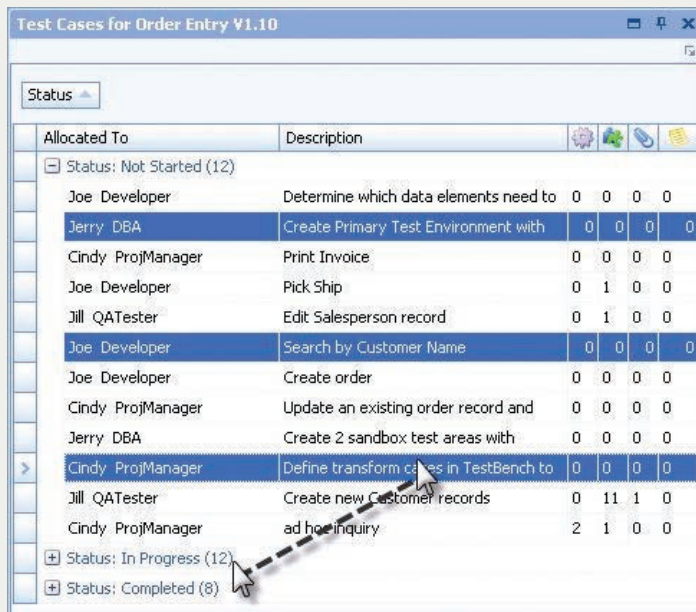


Figure 4: changing status of multiple test cases

the “Allocated To” column heading onto the “Group by” area just above it (figure 2).

Now the affected test cases are selected in the familiar Windows way: click the first and shift-click the last. All are dragged and dropped onto the “Joe Developer” group (figure 3). And that's it.

Another example: a DBA, a developer and their PM, Cindy, start work on new test cases. Cindy changes the status of all three by grouping them by control-clicking them and dragging them to the “In Progress” group (figure 4). They are now tasks in progress throughout Qualify: in Gantt charts, reports, the affected users' calendars (figure 5) and work lists, and the practically infinite other data views.

Auditability and accountability for today's business world

It's no longer enough to report the results of testing. Business stakeholders need the ability to define their own reporting so that they can understand precisely what has been done at whatever level is needed to meet their rapidly-changing needs including evidencing compliance. Qualify provides absolutely detailed history so that complete audit trailing, versioning, rollback and coverage measurement can be achieved easily. It outputs fully customizable reports in various formats including publish-ready HTML and PDF ■

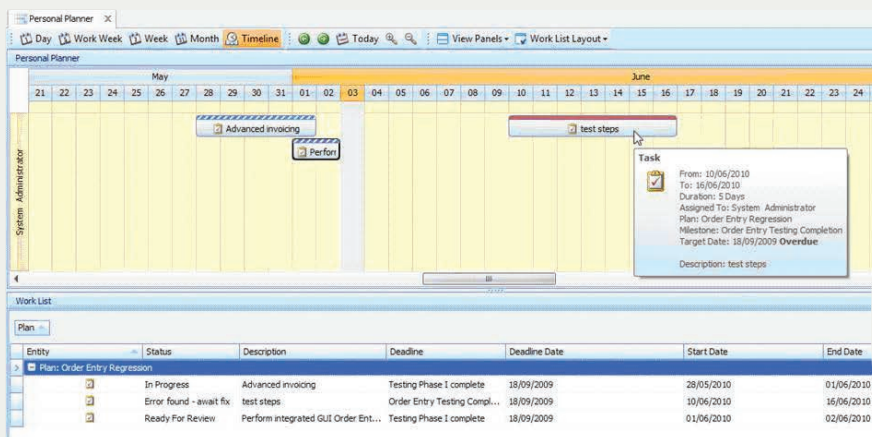


Figure 5: Qualify's planner view

Suppose “Sally Business Analyst” is indisposed and “Joe Developer” is to take responsibility for executing her test cases related to the requirement “Order Entry 1.10”. First we view all test cases for that requirement (figure 1). Next, we group them to get all Sally's ones together: that's done simply by dragging

George Wilson is a founder and general manager of Original Software (<http://origsoft.com>)

Revelation space

Advanced Test Management

by Patrick Hendrickx and Chris Van Bael

ps_testware, ISBN 9789090257273

Available from <http://amazon.fr>, soon from <http://amazon.com>

In his foreword Alain Bultink says that ps_testware and ISTQB share the same philosophy of testing. That is the great strength of this book: the authors have embraced completely the often mysterious *ISTQB Advanced Level Syllabus* and explained their interpretations of it more clearly than anyone else has yet managed. Even better, the explanation is practical: the titles of many sections begin with “How to...” and they really do demonstrate actually doing the things needed to choose the right exam answer. Diagrams (not including the silly photos at the start of each chapter) are well executed. As the

syllabus requires, much is drawn from other sources, but the whole adheres tightly to ISTQB’s prescriptions – exactly what someone aiming to pass *Advanced Level Test Manager* needs. Indeed, the tripartite nature of the syllabus means that portions of the book can be used also by those studying for *Test Analyst* and *Technical Test Analyst*. A subset of the content would also be good, in my opinion better than BCS’s official book, for Foundation Level candidates. The large page count (685) is due to the use of “structured writing” which is granular, organized and formulaic. Paragraphs are

very short with many in tables, and element types and subheadings are plentiful and diverse. It works brilliantly for study and reference, but less well for personal learning and understanding, because the fragmentation makes linear reading heavy going: there is almost no narrative. So, this is a fine study aid, positively essential for anyone taking ISTQB-AL-TM. It’s also a good textbook, although it’s better to dip into than read through, and would be even better if it contained fewer defects: typos etc fall to the eye too readily, and those publishing books for testers should do more to convince readers that they are eating their own dog food. Whether it’s a sourcebook for a test manager depends on to what extent he or she agrees with ISTQB, but it contains much valuable, accessible information and is undeniably a worthy addition to the genre.

The Little TMMi: Objective-Driven Test Process Improvement

by Erik van Veenendaal and Jan Jaap Cannegieter

UTN, ISBN 9789490986032 Available from <http://www.utn.nl>

As contributors to and enthusiasts for TMMi, the authors want it to be adopted by more test organizations. Their book aims to promote that and is deliberately compact in order to make the model more accessible. In fact 50 or so pages are the text of the

model, but with the low-level detail removed: over 56,000 words cut down to about 13,000. That obviously makes it easier to digest, but the very stiff and general style remains. Explaining what it means (which is often far from obvious) in

a less formal way, with examples, may have achieved the objective better. The last 20 pages are original: they describe assessments, then implementation using IDEAL. These sections are much easier to read and are worthwhile, but again are prescriptive rather than instructive. This book is a convenient way to learn what TMMi says we should do, but a practical, self-contained guide to tell us *how* is needed too.

BS 8878:2010 Web accessibility – Code of practice

BSI, ISBN 9780580626548

Available from <http://bsigroup.com>

This new standard has replaced PAS 78:2006 *Guide to good practice in commissioning accessible websites* and is very well written in a modern style making it far more readable than that and other older standards such as those familiar to testers. It’s an essential guide for anyone formulating new web strategy, such as a startup, and more experienced web application managers will appreciate having almost all the current advice in one place. For example, did you

know that a United Nations convention (<http://un.org/disabilities/convention/conventionfull.shtml>) requires “products to be usable by *all* people to the greatest extent *possible*”? Other than mind-boggling but useless facts like that however web testers will find nothing new. We are told to create an accessibility test plan. Its obvious contents are described in four brief bullet points. The test methods mandated are markup validation, WAI conformity checking,

executing tests without using a mouse and with assistive technologies, “expert reviews” (heuristic evaluations and walkthroughs) and observing representative users. Finally we are reminded to repeat accessibility testing when the site is updated. The best information in this document, such as the clear explanation of current legal and other obligations, business justification for good accessibility and discussion of the needs of people with different disabilities has implications for testing but does not help to do it. After all these years, a usability and accessibility testing standard or at least textbook providing innovative, applicable test techniques is still sorely needed.

There are two levers that can be used to improve test effectiveness and efficiency: the one is to impact test coverage and the number of test cases and the other the level of automation. We already gave you an insight into how TOSCA Testsuite™ increases both the effectiveness and efficiency of your functional software testing. To make that happen TOSCA offers the following four unique abilities: ❶ tangible test coverage and traceability, ❷ business dynamic steering that eliminates the maintenance problem of automated tests, ❸ „TOSCA OneWorld“ to unite GUI and non-GUI as well as manual and automated tests and finally ❹ TOSCA@data the solution for generating a dynamic and synthetic test data base - another indispensable prerequisite for high automation in regression testing.

Today: ❷ Finally a solution to the maintenance problem?

The cost-intensive maintenance of scripts and frameworks has always been the problem child and eventually the show stopper of test automation. Functional changes to the system under test require adaptations in the test cases. With conventional script based tools, a great deal of effort must be put into meeting the maintenance requirement. As a result, test case portfolios quickly „die off“, are sometimes no longer executed or are discontinued. TOSCA reduces test case maintenance by up to 90% and therefore solves the maintenance problem: business dynamic steering allows business testers to specify AND maintain test cases without any programming.

Test cases can be automated at any time by drag & drop and TOSCA offers the highest possible degree of user convenience by offering dynamic expressions and auto-completion. Dynamic adjustment at runtime eliminates the bulk of the maintenance workload and TOSCA's modular concept reduces unavoidable maintenance to the conceivable minimum. Dynamic steering allows the test to become what it should have always been: *a business discipline.*



Want further information? Visit us at
www.tosca-testsuite.com



Test library

Selenium Simplified

by Alan Richardson

Compendium Developments, ISBN 9780956733214

Available from <http://www.compendiumdev.co.uk> and <http://amazon.com>

This is a tutorial to be followed by a learner with hands on a computer – the crashiest of courses imaginable, in not only Selenium but Java, XPath, CSS Selectors, JUnit and much more. Its direct, conversational style is fast-paced but very easy to follow, and well-explained code and screen shots are plentiful. It's highly accessible, easily achieving its goal of being suitable for almost anyone, even those with no previous coding or automation knowledge, and will be enjoyed just as much by the experienced. Windows is used throughout, but other than in the short sections on installation etc, the steps are similar under Mac or Linux. Whatever the OS, in parts where the test items and tools interact directly with it some readers might be in danger of

getting stuck if something about their environment causes behaviour different to what the narrative expects. Apart from that, the only weaknesses result from the author's decision to self-publish. That's obviously to be encouraged, but the production is not yet good enough. The A4 format makes it too big and heavy for comfortable reading in the hands, yet it is perfect bound, so will not lie open on a desk, and graphics are printed in greyscale, reducing the readability of screenshots showing code being edited. Finally, no book should be judged by its cover, but especially not this one. Buying the e-book entitles one always to download newer versions, which have already addressed many of the language and layout defects in the printed copy we

reviewed, so is easiest to recommend. Anyway, that's the best way to read it – onscreen, in colour, alongside the applications it teaches you to use. It looks fine on a handheld reader too. Whichever format you prefer, if you want to learn this stuff you must get this book – there's no better way, except being shown by the author in person. Testing, not only automated, needs many more truly practical books like this.

*Thanks to its author **we have one printed and three e-book copies of Selenium Simplified to give away.** For your chance of getting one, send an email to books@professionaltester.com telling us what is your favourite testing book and why. The free books will go to the writers of the first four emails received*



15% off all SQS training
for Professional
Tester readers*

Shoot for success

All your testing requirements firmly under control

Not only does software testing save time and money; it also protects your good reputation. Ensuring high quality in your IT projects is challenging and requires continuous training.

- Specialist training can solve your real-life testing and QA problems. Give your testing team the winning edge with best practices drawn from successful real-life projects!
- Learning how to best use industry-leading tools from vendors including HP and Microsoft will secure you maximise your investment and increase your team's efficiency.
- For practitioners, our ISTQB® Certified Tester training series drives confidence and guarantees a secure platform to build success on.
- Our TrainingFLEX scheme will help you squeeze more value from your training budget and ensure success.

* Courses must be booked and taken by 30 April 2011, quote code PT1102 to qualify

Details of all SQS training services are available at www.sqs-uk.com/training

