PROFESSIONAL TESTER

We're back!

Essential for software testers

January / February 2010 | £8 / €9 | v2.0 | number 1

Very early lifecycle testing

With contributions from:
Barry Weston, Ben Visser *Sogeti,* Yann Gloaguen *SQS-UK,* Erik van Veenendaal *Improve Quality Services*

SOGETI

Introducing TPI® NEXT

Sogeti's Business Driven Test Process Improvement

The world's leading approach for improving test processes... has now been enhanced!

Sogeti's new book, **TPI® NEXT, Business Driven Test Process Improvement** written by Sogeti test experts and validated by extensive customer field tests, builds on the strengths of the successful original model and provides invaluable insight and recommendations on the maturity of an organization's test processes.

What's new about TPI® NEXT?

— Key focus on alignment with business drivers is at the heart of the model

— Reflects Agile and Outsourcing changes in testing environment

— New 'Enablers' to identify the impact on broader Software Development Lifecycle

— Clear maturity levels – Controlled, Efficient and Optimizing

— Easy-to-view representation for business management.

How TPI® NEXT can help your test organization

— Develop a clear improvement path to improve quality, reduce time and save costs

— Ensure processes support the most important business drivers

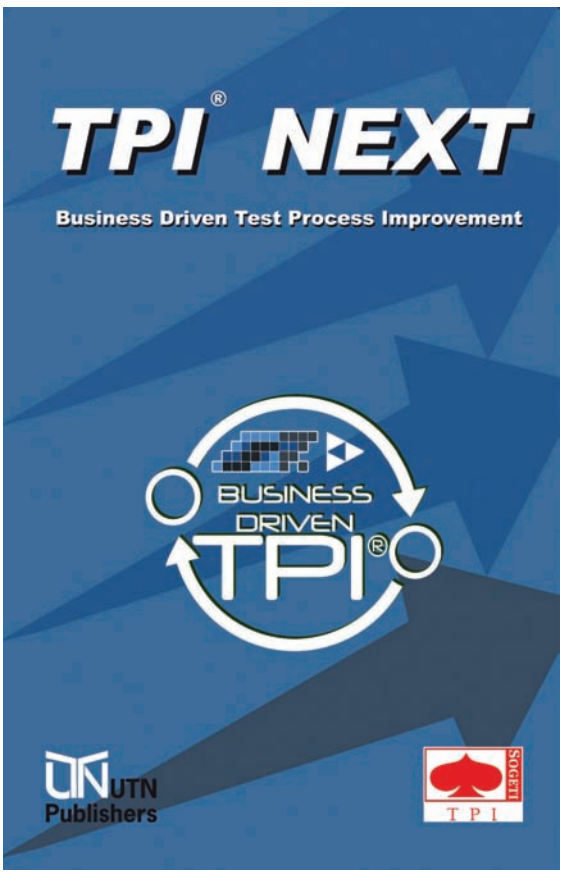
— Gain a better understanding of the correlation between testing and other software development processes.

Order your copy now!

Available from 17 November 2009 from specialist publisher UTN at www.utn.nl or contact Sogeti at tpi@sogeti.nl.

TPI® NEXT an indispensable step-by-step guide to improving your organization's testing processes.

www.sogeti.com



PROFESSIONAL TESTER

Welcome

This is the relaunch issue of Professional Tester! Welcome to readers old and new. We aim to provide practical help and inspiration to testers everywhere, and invite you to be involved.

Very Early Lifecycle Testing

To produce software, it is nearly always necessary to define its requirements. This is true regardless of whether a sequential, iterative, incremental or other lifecycle model, or no model at all, is used.

Sequential lifecycle models tend to rely on producing detailed, rigorous requirements at an early stage as a basis for subsequent products. There is risk that if requirements or constraints change, changing those products will prove very costly.

Other lifecycles – let's use the umbrella term "agile" for convenience – use less detailed and/or formal requirements, allowing the developers to take guesses at what the stakeholders want, and then make adjustments to the developing software as necessary if guesses prove wrong. There is risk that the number and extent of adjustments, and so the time and effort required to make them, will grow very large.

The term lifecycle is borrowed from biology, and like a real lifecycle the software lifecycle begins not with birth but conception (as in life, the conception is usually the more enjoyable). Exactly the same applies to software – but for software, *the concept must be tested.* So how can test activities begin when there is little or no material on which to base them? Can you test what doesn't yet exist?
In this issue we argue that you can and must if danger and waste are to be minimized. VELT is challenging, but is the essence of testing – not just reacting to the work of others, but leading and driving quality improvement.

In the next issue: testing new technologies
In 2009 a revolution began in business IT. Very rapid adoption of cloud computing, virtualization, everything-as-a-service and other newly-enabled concepts will change the ways testing is done – or might make large parts of it obsolete. In the past testers have suffered by failing to keep up with development trends. That can't be allowed to happen again: testers must understand the technologies they will soon be asked to assure.

In the next issue of Professional Tester, we examine the new challenges and share experience of both testing them, and using them to test.

As always, we invite you to email us your ideas and opinions in any form. If you would like to contibute an article please ask for our guidelines and let us know what you have in mind before starting to write.

Edward Bishop
Editor

Contact

Editor
Edward Bishop
editor@professionaltester.com

Managing Director
Niels Valkering
ops@professionaltester.com

Art Director
Christiaan van Heest
art@professionaltester.com

Sales
Rik van Erp
Hakeem Serbouti
advertise@professionaltester.com

Contributors to this issue:
Barry Weston
Ben Visser
Yann Gloaguen
Erik Van Veenendaal

Publisher
Jerome H. Mol
publisher@professionaltester.com

Subscriptions
subscribe@professionaltester.com

Professional Tester is published bimonthly by Test Publishing Ltd.
Email: editor@professionaltester.com.
We aim to promote editorial independence and free debate: views expressed by contributors are not necessarily those of the editor nor of the proprietors.

In this issue

Very early lifecycle testing in TPI® NEXT

Both the original TPI® and the new version TPI® NEXT identify early testing involvement as an indicator of good process. They also promote the practice less directly.

Barry Weston and **Ben Visser** of Sogeti examine what TPI® NEXT, the updated version of Sogeti's Test Process Improvement methodology launched in December 2009, has to say about very early lifecycle testing and find strong support for the assertion that it makes risk reduction more cost-effective.

It's widely accepted that the earlier in the software development lifecycle testing begins, the less expensive the development process becomes.

The V Model as discussed in most training syllabuses suggests that testing should begin when the first draft of business requirements becomes available. But is this early enough? What should "very early" mean at project level?

Being actively involved *before the first requirement has been written* gives testers more influence. The challenge is to make project and development managers see that as a good thing.

Involving testing earlier

In TPI NEXT, the point at which testers become actively involved is dealt with most explicitly in Key Area K02, known as *Degree of involvement* [see panel]. The model provides checkpoints for every key area that are used to assess which of four levels of maturity an organization's test process has achieved, and improvement suggestions to promote compliance with the checkpoints. The lowest level is called *Initial* and indicates an immature process awaiting improvement work. The improvement suggestions for K02 *Degree of involvement* to reach the next level, *Controlled*, include:

Contact both line and project management to emphasize the necessity of (an early) involvement of testing

Start as early as possible with planning the test activities, preferably at:

Project initiation, otherwise at:

Start of the test basis, otherwise at:

Completion of the test basis

So although it might be possible for a process to achieve the Controlled level without VELT, the model indicates that doing more VELT will make that easier and more likely.

VELT to inform stakeholders' risk analysis

Key area K01, *Stakeholder commitment* also has bearing on VELT. For a process to achieve the Controlled level, "stakeholders must commit to and support the test process by granting and delivering negotiated resources". The resources required depend upon product and project risk, and VELT can provide often dramatic assistance here by identifying more risks earlier on. Informing the stakeholders' risk analysis in this way makes the need for resources more apparent and so the negotiation more meaningful.

In particular, detecting inadequacies in whatever requirements information currently exists and communicating the risks those inadequacies pose often

provides opportunities to influence the choice of and secure resources for an effective review process to maximize the chances of finding weaknesses in designs based upon the requirements at the earliest opportunity.

VELT as an integral part of, and to define, test strategy

Key Area K03 is *Test strategy*. At the Controlled level, "prioritization and distribution of the test effort is based on the product risks" - that is, the test strategy is a development from the risk analysis described in K01 and which, as we have seen, is greatly enhanced and made more accurate by VELT.

In describing its significance, the model states that it "aims at detecting the most important defects as early and cheaply as possible". How to decide what is possible is not addressed, and of course there are many defects VELT cannot detect; but the model defines the "most important" defects as those with the highest associated product risks and it is easy to argue, both from intuition and experience, that incomplete or ambiguous requirements are a strong candidate.

The remaining thirteen key areas in TPI NEXT and the richly detailed checkpoints and improvement suggestions contained within them deal with later, more empirical activities which are the only ways of detecting defects in the subsequent development products. But it's clear from the structure of the model - and, we suggest, from experience - that success there is highly dependent on improvement at K01, K02 and K03. If we accept that and consider what is said and implied in these first three key areas, it's hard to avoid the conclusion that TPI NEXT's authors and contributors consider that carrying out more test activities very early is central to making the entire test process more mature ■

K02 *Degree of involvement*

Significance: Tight involvement of testing in the project helps to improve the product quality from the beginning, and helps to keep test activities off the project's critical path: Timely preparation and coordination between different tests can be done and the time that the test project is on the critical path of the project can be kept as short as possible. Early involvement of the test team in the software development lifecycle helps to find defects as soon and easily as possible and perhaps even to prevent errors. At an early stage the test team supports the analysis of project risks and the review of requirements and designs.

From *TPI® NEXT Business Driven Test Process Improvement,*

Barry Weston (barry.weston@sogeti.com) is a senior consultant at Sogeti UK and Ben Visser (ben.visser@sogeti.nl) at Sogeti Netherlands. Ben is a co-author of the books "TPI® NEXT Business Driven Test Process Improvement" *and* "TMap NEXT® Business Driven Test Management" *published by UTN (www.utn.nl). Sogeti Group is a leading provider of professional technology services in Europe, the US and India.*

Towards quantitative governance

Defining the targets that really support management of a test service

When testing functions are outsourced, the various partners and stakeholders often perceive success or otherwise differently.

Offshoring expert **Yann Gloaguen** of SQS-UK explores meaningful test success metrics and how to measure them

Typically, the provider believes its performance is good, testers in the development organization are ambivalent and the parent business is unhappy.

This unevenness is often blamed on ineffective communication. The business says that timelines, specifications and prioritisation have been misunderstood; the provider says they are in fact inappropriately defined. Some suggest addressing this problem by formalizing communication processes, creating more and earlier opportunities to notice discrepancies in understanding. However this can also have negative impact on delivery if forced upon organisations for which it is not a good fit.

Other people believe the main problem is expectations. Early in the engagement definition qualitative assumptions tend to be made, some of which may not be tested until a time significantly after the service delivery begins. When they prove incorrect, surprise followed by dissatisfaction ensues. An obvious and typical solution is to specify key performance indicators and service level agreements in comprehensive contracts, reviewed and signed off by sponsors and lawyers from both sides. However experience shows the success perception gap continues to exist despite this work.

The definition of a metric must include its measurement method

I believe that is because the contracts often fail to detail the measurements to be taken for comparison with the defined targets. “Staff retention”, “capacity buffer”, “productivity”, “defect detection”, “milestones met” and “test coverage” are metrics commonly found in contracts and/or used to govern a testing service. Yet their meaning differs from one organization to another. The ways they are measured can look inappropriate to the unsatisfied; it's easy to get the impression the measurement has been designed to ensure the agreed targets are shown to have been met.

“Staff retention” often contains a “planned/unplanned staff loss” component calculated on a “rolling period”. But what is that? Suppose the target metric is 75% staff retention and there is a team of 40 testers on the 1st January. To some, that means simply that the target is achieved if 30 of them are still in the team at the end of January. But others will adjust for those testers who it is already known will be leaving, and count only unplanned loss. So we need to add a definition of that: for example, is termination of an individual's contract planned or unplanned? And what about project ramp up/ramp down periods, where the requirements for both size and skill profile of the team change? How and when the measurements will be taken and calculations made needs to be defined explicitly, based on risk, remembering that the metric is actually designed to measure not retention of individual people but of the knowledge needed to protect business continuity and efficiency.

Measurement can be intrusive

“Productivity” is one of the most challenging areas to assess. Some use “number of tests executed per person per day”, others “increase in requirements coverage per effort unit”. The truth is productivity is a measure of the value added by individuals over time. Testers know that their contribution is not in executing tests but in understanding and

analyzing change and its impact, understanding risk, making good choices of approach, method and technique, planning and reporting. Business however tends to associate productivity with the execution phase of a project and extrapolate it to other phases, assuming that if the measurement appears to increase during the execution it is certain that we are doing better in planning and design too. That is not necessarily the case, because the act of taking measurements itself impacts individuals and work processes. Calculating productivity based on number of test runs per person per day sometimes causes testers to create more and ever-smaller tests! The faster each test can be executed, the easier it is to achieve the productivity targets. And that's not the end of this syndrome: the smaller the tests the fewer test points in each of them, hence the more tests required to run for a release, supporting a business cases for headcount increase: let's do more, less efficient, higher-cost testing.

So how can we measure productivity meaningfully and in a way that tends to improve rather than worsen actual service delivery? Here is a proposed method.

Productivity has been defined as "a measure relating a quantity or quality of output to the inputs required to produce it" [2]. This poses two questions: "what does testing produce?" and "what does testing require?".

Testing produces quality, by the definition "the totality of features and characteristics of a product or service that bears its ability to satisfy stated or implied needs" [3]. For our purposes, this might be measured as the ratio of number of defects found during testing to the total number of defects found. Testing requires testers, but the number of testers is not helpful to measuring productivity because the individuals acting as testers at any given time will be doing different tasks with varying effectiveness.

It's better to use funding:

$$\text{Productivity of testing} = \frac{\text{number of defects found during testing}}{\text{total number of defects found} \times \text{money spent on testing}}$$

If the number of testers is used, what is being measured is not productivity but efficiency, the as output per worker per unit time [4]. To measure the "efficiency units" spent for a release, we would average the productivity across features, applying weights related to test effort per feature tested:

$$E = \frac{\sum_{i=0}^{n} P(i)T(i)}{\sum_{i=0}^{n} T(i)}$$

Where:

E = efficiency units spent

i = represents the feature tested, a member of the list of n features (eg functions, business processes etc)

P = productivity as defined above

T = the actual test effort

In contracts, "Capacity Utilisation" often goes hand in hand with Productivity. This is most often used for assessing whether a test team is under-staffed. It should be used also for understanding how much effort is spent on different activities.
A "stacked" graph plotting utilization against time (see figure) is a clear way to represent capacity utilization.

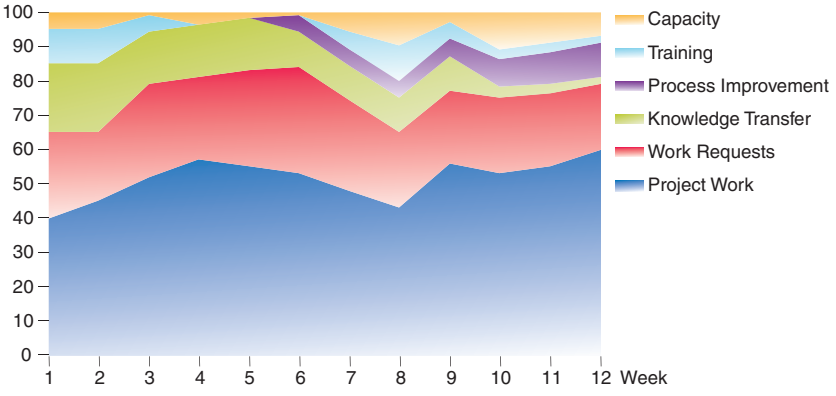
There is little case for hunting down numbers or requiring detailed status reports from the service provider to create these graphs. The objective is to discover a trend for forward planning based on how effort is divided between activities. That should be used not for audit or control, but for governing and giving direction.

Test progress is not the same as quality assurance growth

"Test coverage" is another abused metric. Some calculate test coverage as the number of tests executed divided by the number of tests in scope. Others use test requirements covered divided by total number of test requirements, or perhaps number of test requirements covered by executed tests over the total number of test requirements. But what about coverage of the business requirements? Good testing practice tells us that test requirements must be derived from business requirements and that tests must be derived from test requirements. What degree of confidence will excellent "test coverage" give us when achieving poor "requirements coverage"? When defining metrics, measurements and targets business often will overlook this. It is therefore essential to consult with the QA organization to agree the precise meaning of these vital indicators of quality and progress, and to understand the difference between them.

From the business perspective, "percentage of milestones met" is a metric of paramount importance. This should be a true indicator of expected time to market and reflection of responsiveness to business demand. Discussion of it is often stormy for several reasons: first, the definition of a milestone. At SQS we have seen milestones mistaken for tasks and vice versa many times. *Milestones do not have duration;* they are the product of multiple tasks delivering towards one

Capacity utilisation graph



objective. So “test approach analysis” is not an example of a milestone; “test approach document signed off” is. When tasks are called milestones, we not only weaken and deviate from the plan by omitting dependencies, but increase the number of “milestones”, diluting failures.

This brings us to a second consideration. Which milestones to choose? They should be hand-picked from project plans, selecting the ones that if missed will impact directly on the business. One could argue they all will. Selecting them all however will result in diluting the perceived impact of a missed one which may in fact be far more significant than many others combined. Again testers should be asked to play a key role in the definition of milestones in contract, not just test, documentation.

Conclusion

Communication is rarely to blame for inconsistent perceptions of achievement amongst parties; what is communicated is. Service level agreements and key performance indicators too often focus on metrics identification and targets definition, ignoring means of measurement and so resulting in inconsistency of expectations between participants.

Testing - outsourced or not - needs a standard defining best metrics and measurements. That would empower managers to build better, more flexible and more effective governance models. That can happen only if everyone involved can agree that the measurements are a true measure of performance and the targets correct ones by which delivery can be judged fairly. That would be of benefit to all parties.

This said, metrics do not deliver projects. They cannot predict or solve the specific problems that occur on the ground. Although they are useful in assessing and improving processes, they can never be a substitute for the creative, pragmatic management that comes from experience. That should be the first consideration in any testing-as-a-service engagement ■

[1] http://economics.about.com/od/economicsglossary/g/productivity.htm retrieved 14th December 2009
[2] ISO 8402:1994 Quality management and quality assurance: Vocabulary
[3] http://economics.about.com/library/glossary/bldef-efficiency-units.htm retrieved 14th December 2009

Yann Gloaguen (yann.gloaguen@sqs-uk.com) is the offshore delivery manager at SQS-UK. SQS Group's success in delivering offshore testing has resulted from having highly-qualified personnel both on and offshore, all trained to use SQS' Global Offshore Delivery Model which provides proven measurements guaranteeing continuous visibility.

VELT for Web [part one of two]

How to detect defects in non-existent requirements

Testing is powerless against the worst defect a test basis can have: incompleteness.

PT editor
Edward Bishop on the key challenge of VELT: making something testable from nothing

Software testers specialize in finding things that can be improved in products and processes, including their own work. So they tend to have a lot to complain about. And probably the most popular complaint among testers is of inadequate information on which to base test design very early in the lifecycle.

So we want better requirements. But what does that mean? What would our ideal requirements document be like? Clarity, unambiguity and correctness are all desirable of course, and the entire test process is designed to detect defects in these. Something wrong with a document or other product used as test basis (an “error of commission”) will hopefully be detected by verification, most likely during early test design, or at worst by test execution.

But testing in any form is powerless against the worst defect a test basis can have: incompleteness. As the first product of development, requirements can't be validated because there's no earlier document against which to validate them. So something missing (an “error of omission”) is far harder to detect. If it's not there you can't review it and no tests will be created relating to it. In a sequential development lifecycle, it will then also be missing from subsequent work products based upon the requirements, for example designs, which will likely be fatally flawed throughout as a result. Because these contain information more detailed than requirements, they are easier to use as a test basis, but it's now too late; the design is wrong, the tests will be wrong, and the delivered software will have defects - probably severe - testing cannot detect.

Think back to the last few times you tried to get information or carry out transactions on commercial websites, if the memory is not too painful. The majority of users - even proficient IT users - experience failure constantly. There is no way to monitor functional or usability failure in production, and therefore no way to measure its cost - but I believe it is immense. A large proportion of current web applications, even those operated by major organizations, are unfit for purpose because intended change to them is not properly documented and therefore not adequately tested.

So, is it realistic to expect complete requirements? For some software products arguably yes. But for web applications, unfortunately not. Nobody can specify a site, or an addition to an existing site, in sufficient detail, in advance - no person or group is that clever. The design aspects of the requirements will emerge from the development, as understanding of complex functionality, data, and in particular how to make navigation and data entry intuitive, grows. For web (and some other modern development environments) that's the only sensible way for designers and developers to work.

Trying to guess at designs in advance is futile and damaging. Mocked-up pictures of screens can help business and developers envisage things approximately but have no value as a basis for early test design: they don't contain nearly enough information and are open to multiple interpretations. Worse, they are invalid; the real product will begin to deviate from them the moment actual design work begins. *Web requirements should be written, never pictorial*. If a document contains a picture of the product it describes, it's a design, not requirements, and of no use for VELT. Anyone tasked with capturing requirements should be capable of expressing them in writing. If not, they probably don't really know what the requirements are.

The kind of requirements written by business analysts are usually closer in form to what testers need. They should describe what the application or change should enable the different user groups to do, and the business procedures and rules that enable test analysis to derive expected outputs. But they don't address other fundamental properties which are crucial to the business success of a public-facing website - the characteristics the site should and should not possess, especially (but not only) in its presentational layer and user interface. That causes shortcomings, inconsistencies and deviations from prevailing standards to be noticed during or after functional test execution, leading to uncontrolled, untested, very risky change too late in the delivery process which is the main cause of rotten websites going into production.

So working within these limitations and being realistic, how can testers get all the important aspects of the site on the test path? The answer is to document them ourselves as part of the test design activity. Most of the design decisions affecting the new functions will already be known to us, or easily predictable. When that is not the

case, we can write them generically; specific information can be added when available and if necessary.

The deliverable from this activity can be called "test objectives". They define what testing will aim to achieve, and can also act as the "title" of a test script or other testing activity during test planning, making traceability and prioritization easier.

More importantly, in the process of creating the test objectives we review the business and other requirements that are available, detecting defects. By considering those requirements systematically with reference to the fundamental functions of any public-facing website, we add related test objectives which would otherwise be omitted.

The review component of this activity can be made more powerful by enforcing a rule: that test objectives must be achievable. This means absolute language such as "to prove", "always", "never" etc cannot be used, because testing cannot show that something will always, or never, happen. Taking care with the wording in this way often reveals ambiguity in the requirements and raises useful and important questions needing clarification. So creating test objectives can detect not only errors of omission in requirements, but other potentially severe defects too, earlier than they can be detected by test design.

Once we have our best effort at what we consider to be a complete and theoretically (not practically - that will be decided during prioritization and later, estimation) achievable set of test objectives for the new functionality, we can use them as a basis for creating and detailing test cases, which should reveal if anything important is still missing ■

The final part of this article presents a fully worked, practical example of applying this method to a real business requirement. It will appear in the March issue of PT, or is available to read now, free at professionaltester.com.

Edward Bishop is author of many training courses on web testing

Incident Log

End of browser war could mean hell or heaven for testers

Some time before 17th March 2010, Microsoft will issue a Windows Update recommended for all PCs running Windows XP, Vista and 7 throughout Europe designed to make it very easy for Internet Explorer users to turn all IE functionality off and install an alternative browser. A "ballot screen" will appear offering twelve brands, including all the major and some less well known ones. It's part of Microsoft's response to antitrust charges issued against it by EU regulators at the beginning of 2009. For full details see http://computerworld.com/s/article/9142416 (retrieved 18th December 2009).

So, you've tested your public-facing web application in a benchmark environment and then performed compatibility testing on the browsers most of your users prefer? Think again!

This doesn't affect only web testers. The browser is the enabling client-side technology for most cloud service models, now in massive growth and likely to be ubiquitous by the end of 2010. Nearly all software organizations will be heavily involved in that.

So, what to do? Until a good proportion of users have seen the ballot screen, data from HTTP logs is useless. Even after that things may not settle down: popularity of browsers is likely to ebb and flow, driven by incremental improvements and word of mouth. Guessing is very dangerous. The only near-certain outcome is a more even and more fluid spread of installed base between more brands. Organizations should be looking now for new ways to assure functionality on that wider range.

Most testing budgets won't allow thorough empirical testing on a lot more client environments. But not doing it is likely to give rise to failure near or after deployment. That means more late fixes and therefore late regression, and that's a leading cause of testing being degraded from the well-integrated, highly-skilled, process-driven discipline it should be to late-lifecycle tail-chasing bug hunting that anyone can do.

On the brighter side, if more organizations opt for analytical compatibility testing, coding standards and the methods and tools used to enforce them may be strengthened. In that case, the applications they produce will tend to work best on standards-compliant browsers, influencing more users to choose them, which in turn will require more applications to become more compliant... this won't get rid of abominations such as IE6 overnight but it would definitely speed the process up. Instances of standards successfully improving matters for the IT industry and the users it serves are rare as hen's teeth. Thanks to Brussels, could we be on the verge of a great one at last?

Instances of standards improving matters are rare as hen's teeth

Improving process improvement

To many, test process improvement means following one of the two best known reference models: TMM (relaunched as TMMi in 2005) and TPI® (relaunched in late 2009 as TPI® NEXT and discussed in this issue of Professional Tester). But there is more to TPI than these.

Erik van Veenendaal introduces some of the many approaches to test process improvement.

The ISTQB Certified Tester Expert Level syllabus of which I am an author, due out in early 2010, is entitled *Improving the Testing Process.* That fact reflects the importance placed on the subject by testing thought leaders and their organizations which face ever-increasing challenges.

There are various angles from which to work and process-orientated seems to get the most attention. Others such as people-oriented (make sure your testers are top of the class and they will do a better job) or automation-oriented (use a unit-test framework and/or automate regression tests) are also proven ways to improve testing. Don't focus only on process: balance your improvement efforts!

The syllabus covers TMMi and TPI® in detail, but also other process-based approaches. Here is a selection with references and brief explanations.

IDEAL [1]
An organizational improvement model that serves as a roadmap for initiating, planning, and implementing improvement actions. The IDEAL model is named for the five phases it describes: initiating, diagnosing, establishing, acting and learning.

Fundamental Concepts of Excellence [2]
Nine criteria: results orientation, customer focus, leadership and constancy of purpose, management by processes and facts, people development and involvement, continuous learning, innovation and improvement, partnership development and corporate social responsibility.

Critical Testing Process [3]
A content-based model for test process improvement built around twelve critical processes. These are highly visible: peers and management judge competence and mission-critical operations where performance affects profit and reputation. The model is context sensitive and allows itself to be adapted, including in identification of specific challenges, recognition of attributes of good processes and selection of the order and importance of implementation of process improvements.

STEP (Systematic Test and Evaluation Process) [4]
A structured testing methodology, also used as a content-based reference model for improving the testing process. Does not require that improvements occur in a specific order. Seven basic premises: requirements-based testing strategy, testing starts at the beginning of the lifecycle, tests are used as requirements and usage models, testware design leads software design, defects are detected earlier or prevented altogether, defects are systematically analyzed, testers and developers work together.

Cause-Effect diagrams (also called Ishikawa fishbone diagrams) [5]
A brainstorming technique to identify clusters of causes and symptoms whose solution will provide the most benefit. It uses graphical representation to organize and display the interrelationships of various possible root causes of a problem. Possible causes of a real or potential defect or failure are organized in categories and subcategories in a

horizontal tree structure, with the (potential) defect or failure as the root node.

Causal analysis during inspection process [6]
A review technique to facilitate analysis of the causes of detected defects and identification of actions to eliminate them.

Goal-Question-Metric (GQM) [7]
An approach to software measurement using three levels: conceptual level (goal), operational level (question) and quantitative level (metric).

Fundamental change management process [8]
A structured approach to transitioning individuals, teams, and organizations from a current state to a desired future state in eight steps: create a sense of urgency, pull together the guiding team, develop the change vision and strategy, communicate for understanding and buy-in, empower others to act, produce short-term wins, don't let up, create a new culture ■

[1] IDEAL: A User's Guide for Software Process Improvement, McFeeley. SEI 1996, CMU/SEI-96-HB-001
[2] efqm.org
[3] Critical Testing Processes, Black. Addison Wesley 2003, ISBN 0-201-74868-1
[4] Systematic Software Testing, Craig and Jaskiel. Artech House 2002, ISBN 1-580-53508-9
[5] Problem Solving in Groups, Robson. Gower, ISBN 0-566-07415-x
[6] Software Inspection, Gilb and Graham. Addison Wesley 1993, ISBN 0-201-63181-4
[7] Software Modeling and Measurement: The Goal Question Metric Paradigm, Basili. University of Maryland 1992, CS-TR-2956 (UMIACS-TR-92-96)
[8] Our Iceberg is Melting, Kotter and Rathgeber. Pan Macmillan 2005, ISBN 978-0-230-01420-6

Erik van Veenendaal (eve@improveqs.nl) is founder and director of testing consultancy and training provider Improve Quality Services Ltd, former vice-president of ISTQB, vice-chair of the TMMi Foundation and author of many books on testing. The beta version of the ISTQB Certified Tester Expert Level syllabus is available at istqb.org.

Seeking the truth behind legendary failures

Everyone in testing has heard, and many have retold, stories of software product and/or project disasters. They are bandied about particularly often in training situations, when course presenters use them to explain their points, or sometimes just to reinforce the importance of being aware of the risks of unexpected failure.

Some classic cases involving space vehicles and dangerous medical machinery have been well documented and analyzed over decades (but still not well enough to make it obvious what lessons modern business software testers should learn from them). More recent ones are probably partly and possibly wholly apocryphal.

Nevertheless, they continue to be retold, apparently as fact, by people with no actual evidence or first-hand knowledge of what happened. Rude Coarse Analysis is where you can help Professional Tester put a stop to that and discover the real truth.

To get the ball rolling, here are three old stories we, and probably many of you, have heard in the last few years - in many cases, from people who were (or may have been) working close to the actual events. Professional Tester's purpose in retelling them is research: ***they may well be partly or wholly untrue.*** If you know that to be the case, help us correct the record. We want to help find out what really happened and what, if anything, testers can learn from it.

1. Nectar launch fiasco

When Nectar, the retail loyalty points card used by customers of BP, Sainsbury's and others, was launched a key message of the blanket advertising was “sign up online and get free points”; persuading consumers to do this was considered desirable because (i) getting the customer to enter their personal data is cheaper than doing it yourself, and (ii) it created a chance to generate repeat web visits by cardholders, with cross-selling opportunities.

But there was a compatibility failure. Some users completed signup with no problems, but clients running Windows 2000 and Internet Explorer 6 caused a session to be created that could not be dropped (details unknown). The servers ran out of memory, file handles or some other resource very quickly and the site became unavailable to all users.

The users didn't want to miss out on their free points so a lot of them phoned in (a number was publicized alongside the URL). Nectar had to hire half of Mumbai to staff emergency call centres whose function was to give away free money at great cost. The loss of cash must add up to a frightening figure indeed. The loss of potential customers, reputation and partner confidence is worse.

Testing had been prioritized by risk. An extensive risk catalogue was created and scored. Risks involving loss of money tended to score highest: eg “a customer is able to register for an offer requiring more points than that customer has accrued”. Others that involved large partner organizations scored high too, especially on likelihood, because the functions concerned required complex interfacing to the partners' systems: eg “points earned by an EDF Energy customer when paying an electricity bill online are not credited to the customer's Nectar account”.

But, although these are high-scoring risks, *they were actually very unlikely to happen early in the life of the site when hardly anyone had yet signed up.* This is a common mistake in risk analysis. We tend to consider the likelihood of a defect being present: that's different to the likelihood of it causing a failure. That depends on the actions of users, which are usually unpredictable. But in this case they - or rather the actions most users could not take, because they didn't have an account yet - were obvious.

So, the prioritization was wrong. More resources should have been dedicated to testing the signup process before and after the launch, and fewer to other functions, until a specific proportion of the expected registrations had been completed. Risks such as “a particular combination of client platform and browser causes exponential growth in server resource consumption” looked relatively unlikely, but when considered in terms of the signup functions, to which high traffic had been driven deliberately by advertising, it was far more likely than risks to other functions to which very few users yet had access.

Nectar's explanation at the time was that it had been surprised by the traffic levels. This is spin: dressing up a disaster to look like popularity, by saying “we are so great we couldn't cope with demand”. Naturally commentary on the event was immediately taken up also by marketers pushing performance testing tools. If used correctly, ie to simulate multiple client configurations, and with more luck, some of these might have detected the defect (the planned performance testing was in fact delivered successfully using various tools). But really it should have been detected by functional integration or system testing, and would have been if critical tests had been executed on common client environments - but the risk analysis said that wasn't necessary.

2. NHS patient records system mistake

This time the culprit was not risk analysis, but risk identification.

It's easy to criticize some NHS IT initiatives, but the need for this product was clear to all. A patient's medical records, including imaging and test results, would be available to medical professionals, as appropriate and authorized, at any NHS site the patient attended. There would be no waiting for information from GP surgeries or other institutions or departments. Efficiency,

quality and timeliness of treatment would all improve. Lives would be saved.

The success or otherwise of the project was important for other reasons too, including to the government which had authorized the huge investment. So at the appropriate point in the project, a risk analysis meeting was held at 10 Downing Street with Prime Minister Tony Blair participating. The risk catalogue was considered and scored by consensus; apparently Blair tended to extreme caution and scored nearly everything the maximum 10.

Crucially however, one risk which later turned into reality was missing from the catalogue: "medical professionals who should use the system will not accept it". To be fair to Blair, there is no implication that he was made aware of this risk, had any opportunity to envisage it himself, or could reasonably have been expected to do so.

The system was built and tested and proved dependable. It retrieved and displayed records effectively. But updating records after seeing a patient was unwieldy, requiring a lot of navigation to complete all necessary fields. Doctors and nurses found this took an average of 7 minutes. The time allocated to a general practice appointment was 10 minutes, so this caused intolerable delay. Subsequent patients were seen late and this got worse as the day went on.

The medical professionals were forced to go back to making written notes and instructed their administration staff that these must continue be available at patient appointments. Result: a £1.3bn system into which nobody was entering any data. Lesson: beta testing is too late to detect acceptance issues, and acceptance testing is too late to detect design defects. Static testing should have prevented this loss approximately one year earlier.

3. UK National Lottery whitewash

Before it became possible to buy tickets from the National Lottery website [national-lottery.co.uk], players bought them from retail outlets, where the operator Camelot installed around 25,000 specially-designed terminals. The terminals include a scanner for optical mark recognition which inputs the chosen numbers from a "playslip" completed by the player. This saves time and avoids claims that the operator keyed wrong numbers: it is the player's responsibility to check that the numbers on the ticket printed by the terminal match those marked on the playslip.

When a ticket is purchased, the chosen numbers are transmitted immediately to a central database. Among other things, this allows accurate estimates of the jackpot to be made prior to the draw, and the number and geographical location of big winners soon after. Releasing this exciting information is a key part of the marketing strategy and is also required for compliance with the terms of the operator's licence issued by the government; immediacy and openness helps to ensure and reassure that no cheating etc is possible. The terminals work well; we know of only one serious reported failure and that may have been due to network issues beyond Camelot's control. [http://www.guardian.co.uk/uk/2008/dec/28/lottery-camelot, retrieved 16th October 2009]

The website is good too; in design and usability at least it may be one of the UK's best. Selling lottery tickets online is very desirable because they, like many gambling purchases, are time dependent. The customer has a strictly limited time in which to buy, so offering additional and convenient channels is vital. There are obvious risks too, including high demand as the end of the buying window (before the draw) approaches. Camelot operates in a strictly regulated environment so development and testing work was carried out to a high standard and a clever design that makes it easy for almost anyone to buy and pay for tickets, and reduces peak traffic through identified bottlenecks such as the merchant-bank interface used to accept debit card payments, was implemented.

But one requirement proved unexpectedly difficult to deliver. The data on numbers chosen by online players could not be merged easily with that from terminal sales. The schema used by the terminals system was complex and not well understood except by its producers, whose work had been signed off months earlier. DBAs working on the website back end, a much simpler schema built in a different DBMS, were intimidated. They dared not risk any operation which might cause integrity failure - the slightest suspicion of that would have forced canceling the draw among other terrible consequences. Delaying deployment was an unattractive option too.

There was only one thing for it: deploy on schedule then take the tickets purchased online and enter them into the legacy system. At first this was done manually, by reading the numbers for a ticket off screen, completing a playslip, and processing it just as a newsagent would. In time, ways were found to achieve more automation: a line printer rigged to mark the playslips, then a laptop hardwired to the terminal, emulating its scanner.

But all this took time. Tickets were being sold faster than they could be entered. So the sales function was taken offline from 10pm until 10am every day to allow the data entry operators to catch up and statistics to be produced that were accurate, at least at 10am. Website providers often call this a "planned outage" and try to find ways to exclude it from service level calculations. Everyone else calls it availability failure.

During these 12-hour periods of availability failure an announcement posted on the website gave the impression that all was normal. When asked for details, Camelot spokespeople intimated that the closure was part of its responsible gambling policy: apparently more customers tended to make poorly-judged spending decisions at night or in the early morning. However the nightly closure is now much shorter: tickets can currently be purchased between 8am and 2am. Perhaps further research revealed the public's "window of irresponsibility" was shorter than previously thought. Or a later

technical solution simply removed the need for that justification.
Interfacing old and new systems is often challenging, but in this case the "legacy" system was very new itself. It would be fascinating to know if requirements for that system addressed the predictable need to import or preferably export mission-critical data reliably and safely, how the problem was discovered, and why both functional (including analysis of data to check outcomes following test execution) and non-functional (scalability, interoperability) testing failed to do so earlier ■

Were you there? Do you know what really happened? Or just have an opinion on the credibility of these accounts? Send an email to editor@professionaltester.com. There's no need to reveal your identity, and if you do we won't publish it unless you ask us to.

Agile Testing

by Lisa Crispin and Janet Gregory
Addison Wesley, ISBN 978-0-3215-3446-0

This book is very well written. Its friendly conversational style is highly readable yet dense: there is new information in nearly every sentence. The frequent stories of experience are believable and relevant. Panels and figures are used liberally and to good effect. Theoretical and people-management aspects are covered but the bulk of the text is practical advice and examples, including explanations of a range of tools which more technical testers will find very interesting. Its best attribute however is the infectious enthusiasm of the authors: they clearly love their work and want to share that enjoyment with the reader. Little wonder, because they appear to have complete freedom to experiment with innovative solutions to ever-changing challenges. Deadlines, budgets and governance simply don't come into it, and "risk analysis" means vague guessing of a few things that might go wrong.

So if you are using or attracted to agile development and finding corresponding test approaches, and want an evangelical book with real substance, this is it. If you are more sceptical, like me, it won't change your mind but its persuasiveness may disturb you. I found myself worrying about who pays for all this fun.

How We Test Software at Microsoft

by Alan Page et al
Microsoft Press, ISBN 978-0-7356-2425-2

A bold title that might make one think it's a "shortest books" joke. But no, it's a serious 405-pager. In fact the title is misleading; the content is more about software testing in general than insight into how Microsoft works. It starts from the basics and introduces all the usual methods and techniques. Much of this has been done better in other books, but it's always good to have another explanation, and the examples using actual Microsoft products, although simple, are useful.

However testers looking for new ideas on process will be disappointed. This is very much US-style context-driven late-lifecycle testing. Everything, even application of functional test techniques, starts from the assumption that the test item has already been coded. The word "requirements" is not in the index: I found it only once, in a diagram used only to dismiss the waterfall model and in which "maintenance" is misspelled. The V model is not mentioned; spiral/agile is the only development lifecycle given even a cursory explanation.

The authors aim for a very accessible style but succeed less well than Crispin and Gregory. They use too many irrelevant analogies and insights into their personal lives which are boring (the insights, not their lives). The stories of events within Microsoft do interest, but the picture they paint is one of chaos. The developers are firmly in the driving seat and the testers are really developers. Example test cases are unrepeatable, incident reports are laughably informal, and dubious metrics are of bug creation and resolution, not detection. Surprisingly, issues are assigned to individual developers by the tester raising them: at one point it's suggested that perhaps it would be a good idea if each developer with ten or more bugs against his or her name was asked to fix them all before continuing work on new features. Er, yes! Perhaps Microsoft should scale that principle organizationally, and fix ten faulty products already sold before launching each new one.

TPI® NEXT

by Alexander van Ewijk et al
Sogeti/UTN, ISBN 978-9-0721-9497-8

This review is of the book, not the model itself. That can be evaluated adequately only in practice and over years, as has the original TPI® published in 1998. The experiences and opinions of the many organizations which used it have influenced the new version. So even if you have considered TPI® before, now is the time to look at it and perhaps try it again.

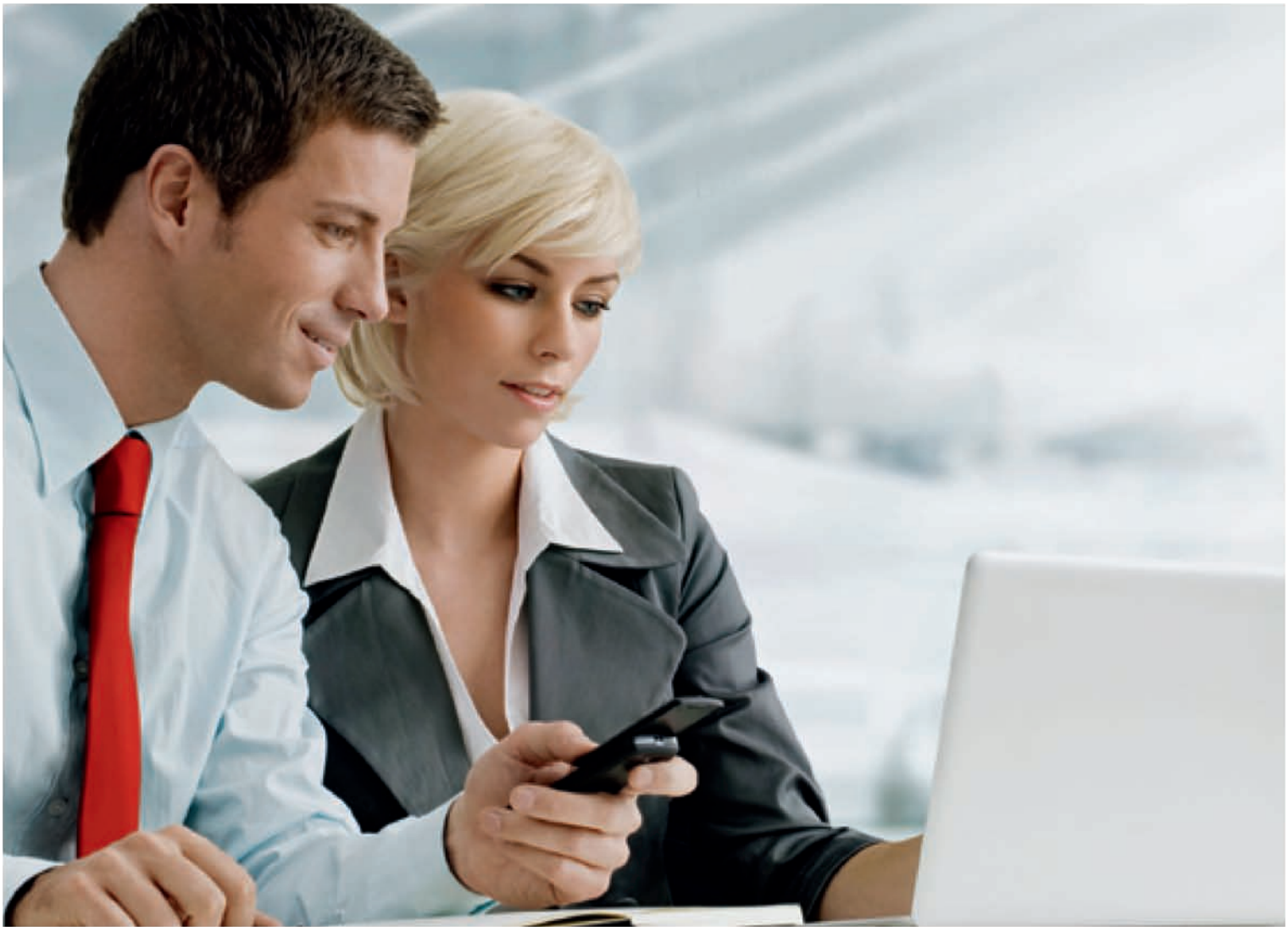
The advice given in the original model appealed to testers but was hard to justify to business. The new one addresses that by relating improvements to testing directly to high-level "business drivers" such as time to market, cost reduction, transparency and so on. Other new sections attempt to make the model more applicable, showing how it can be used with more general quality and process improvement models such as CMMI, in service-based testing situations, and (less convincingly) with iterative/agile development lifecycles.

This book, the definitive reference to the model, is admirably clear and explicit. None of the seven authors has English as their first language and that may have been a good thing; the end product of the translation and editing process is short sentences that get directly to the point with few words wasted and nothing hidden in ambiguity. Typographical and punctuation defects remain and should be fixed, but I found none that affect the sense. Criticisms of the model, including that it is insufficiently rigorous and too open to varied interpretation, may still be valid, but the book is hard to fault.

SWISS TESTING DAY

www.SwissTestingDay.ch

Swiss Testing Day



17 March 2010 - Kongresshaus, Zurich

Details available on the website www.SwissTestingDay.ch

Platinum Partners:

Cognizant

hp

IBM

Check Gold Partners online!