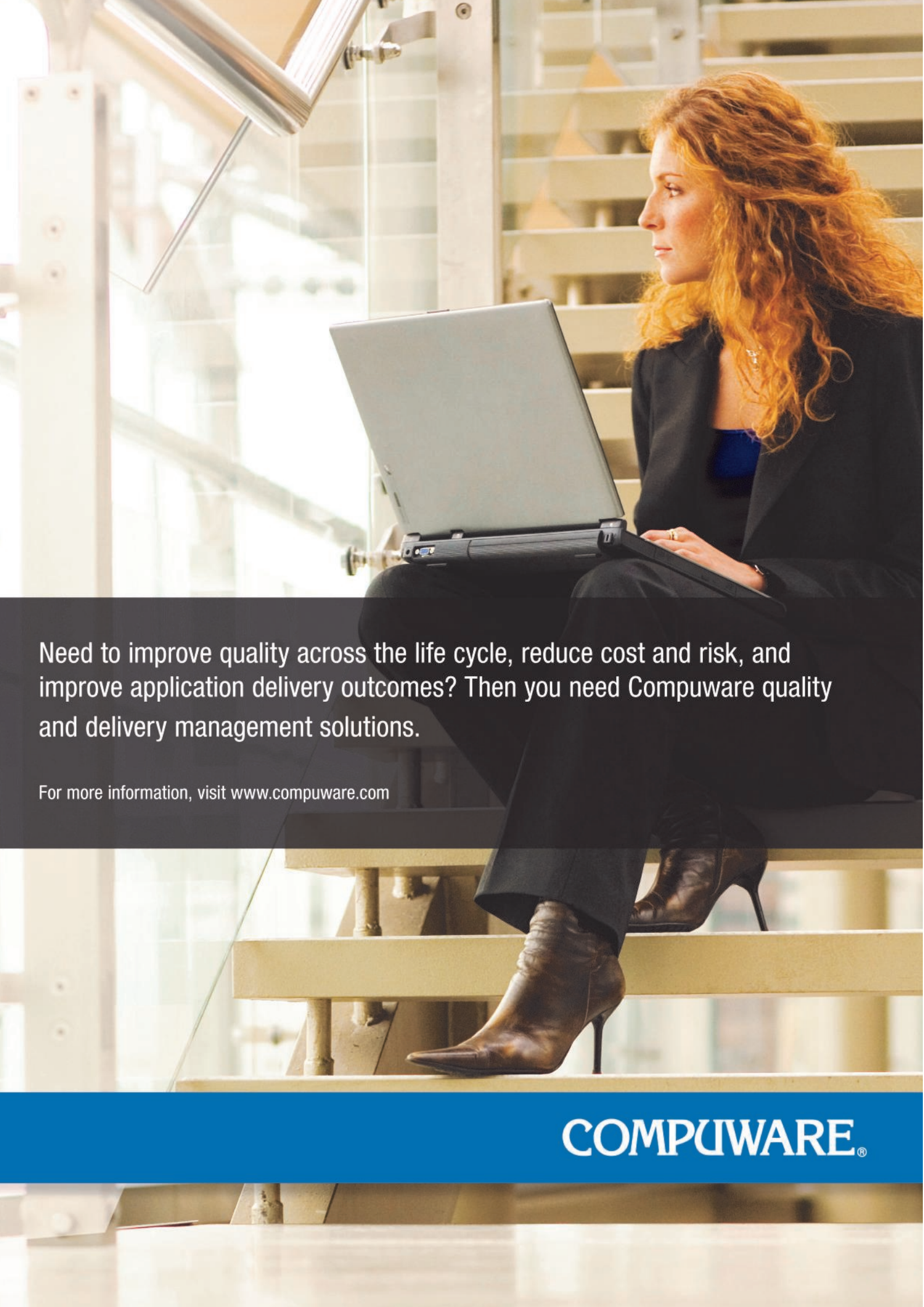
No. 01/08
March 15th, 2008

testing experience

The Magazine for Professional Testers

printed in Germany
free exemplar
www.testingexperience.com

Testing & People


Need to improve quality across the life cycle, reduce cost and risk, and improve application delivery outcomes? Then you need Compuware quality and delivery management solutions.
For more information, visit www.compuware.com
COMPUWARE®

Editorial

Oscar Wilde said “The only way to get rid of temptation is to yield to it”. We have for a long time hatched the idea of a magazine covering the daily experiences of testers and providing a platform for test professionals on the topics and trends around their fields of interest. “testing experience” is intended to be both: a magazine and a platform for discussion. “testing experience” has one mother – we are the mother, a lot of fathers - the authors, and hopefully a big family – all of you! Together, we are the ones who can make “testing experience” grow from a baby to an adult serving as a medium to transfer experience, opinions and trends.

“testing experience” is the challenge of producing a high-quality magazine for professional testers made by and issued for people involved in testing. The real challenge in this kind of venture is to find the right authors with the right themes, the right audience and, last but not least, the supporting community. In the first edition, we rely on a lot of experienced and well-known authors with attention-grabbing articles. We have the support of global as well as small companies advertising their products and services.

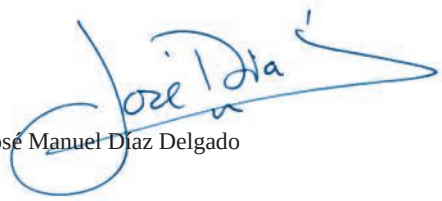
We hope to get the support of the global testing community by asking them to subscribe to the magazine and to make the magazine’s link known to their contacts.

We are open to authors from all around the world. If you have a good subject in the field of testing, please don’t hesitate to contact the editorial staff.

We’d like to ask you to send us your opinion about this first edition and of course about the topics we address. Your opinion is very important to us.

Last but not least, we’d like to thank the organisations and companies that have given substantial support to our magazine “testing experience”.

Yours sincerely



José Manuel Díaz Delgado

Content

18

The Future Tester – What is necessary to know and track?

By Alon Linetzki

A Quality Manifesto

By Tom Gilb

11

Adopt Your Local Professor: The Need for Industry and Academia to Work Together

By Patricia A. McQuaid, Ph.D.

6

© Alexandra Bucurescu / www.pixelio.de

What a Tester Should Know, At Any Time, Even After Midnight

By Hans Schaefer

40

© Gerd Altmann / www.pixelio.de

Adopt Your Local Professor: The Need for Industry and Academia to Work Together

By Patricia A. McQuaid, Ph.D.

All too often while speaking at conferences, I hear people from industry saying that "the students coming out of college these days do not have the skills we need...". While that often is true, I propose a partial solution to this problem that is actually quite straightforward. Simply stated, why not help out and adopt your local professor?

One element that universities and businesses have in common is budget constraints, especially in today's economy. Just like many businesses have cut back drastically with discretionary spending, so have many universities, perhaps even more so. As we know, professional conferences are very expensive; so are specialized training courses in whatever your field of expertise may be.

Now, in computer science and other information technology-related fields, the body of knowledge we need to know is extremely dynamic and very quickly becomes outdated. The technology used now, both hardware and software, was not even invented when many of us professors were in graduate school. So, much of what we teach in our current curricula is mostly self-taught. If we are lucky, we have been able to attend a training course on the topic we are teaching, but more likely, we had to learn it on our own.

So how can dedicated faculty get the training they need to properly educate the future software testing/ software engineering professionals? By having industry establish more partnerships with universities, both informally and formally.

Informal Alliance

Industry spends a great deal of money on training their personnel, from paying for a formal college education, to sending employees to conferences and to professional development seminars. Many of the seminars are held on-site, in which case either consultants are brought in to provide the training classes or they are held by the company's own staff. Very often consultants charge on a per-person basis, but they also may charge based on a range of people, for example a fixed fee for between 20 and 25 attendees.

It is this last case that I would like to address here. How about the next time you have an on-site training class, you invite your local professor to attend at no charge? What better way to help ensure that the college you recruit from teaches the material that you want your future employees to know? What would it cost you? Another set of training materials and a lunch? Or if they do charge by the person, is it worth it to you to pay this incremental cost of educating a professor? Probably so.

Formal Alliance

Establishing a more formal alliance is another option that may be even more worthwhile for everyone involved. This type of an alliance can take numerous forms, including funding a trip for a professor to a conference or for a course, to having corporate personnel guest-lecture at the university, to providing tours of their company to faculty and students, to having the faculty work at their company for a period of time. Providing material and examples that professors can use in their courses is also extremely valuable. One of my challenges is developing real-world examples and exercises that are both meaningful and challenging to the students. You could remove anything confidential, but provide information that we can incorporate into our classes.

Many companies have formal programs that involve professors, from establishing visiting positions ranging from a several month appointment to lasting several years. I was fortunate to be involved in such an alliance several years ago, when I spent five weeks during the summer as part of a major organization's faculty partnership program, working in one of their major software testing labs. This experience was invaluable.

Benefits to the Faculty and to Industry

At the time, I was considering offering a new class for the university, a class in software testing. As part of the preparation for the class, I wanted to learn current practices and to become familiar with some of the automated testing tools that were being used in industry. I wanted to know what was it like being a full-time tester – what challenges do testers face, which automated tools are being used, what are the benefits and limitations of these tools, what is it like being a project manager of a testing team? I wanted to learn

how the organization integrates the tools into their software testing business processes. I also wanted to strengthen the relationship between my university and the organization. I could read the books, but wanted to know more, to improve my understanding and to bring this knowledge into the classroom.

The broad goal for the organization was to have a long-term positive impact on recruiting. More specifically, they wanted to

strengthen ties with my university, as well as to show me how they test software – for their own internal needs and for their clients. They, along with many other companies, were having a difficult time not only locating students that had a formal background in testing, but that even thought of software testing as a career option.

I Was Adopted…

As part of their faculty partnership program, I was able to interview and observe the testers in their daily work. I was also able to discuss issues with software testing project managers. One of the highlights was that I spent several hours learning about usability testing from their usability specialist. An unexpected benefit was that he taped me in their usability lab performing a usability test on a web site, and I use the tape in my classes to illustrate the process of a formal usability test. I also attended several training classes to learn to use two vendors' testing software.

As a result of the alliance, I was able to obtain the training I needed to prepare to offer the new course. In addition to the technical training I received, I obtained a better understanding of current business testing challenges. I offered the new class the following term, with the class comprised of students both from information systems in the college of business and from computer science in the college of engineering. They worked in teams and I made sure that two students were from information systems (business) and the other two students were from computer science (engineering). The students from both colleges had to work together as a team.

The organization benefited by strengthening their alliance with my university and increasing their recruiting success through better student awareness of testing-related careers. Students then considered software testing as a career choice. Potential hires became more suited to the organization's needs, with students obtaining offers for summer internships and for full-time positions upon graduation.

I Now Have New Relatives…

During my stay in industry, one of the managers contacted the chief operating officer at one of the leading providers of automated software testing tools, to ask them to donate software to my university. By doing so, everyone involved would benefit: the university would receive top-of-the-line testing tools to use in my course, the organization would be able to hire recruits that have experience with automated testing tools, and the provider of the automated testing tools would be getting their software in the hands of future software testing decision makers. Meanwhile, independently, the software company began to establish a formal program with additional universities, using my work as a model.

After working with this organization for about a year, they announced their new program, a program designed to provide software and training materials to assist academic institutions to develop their technical curricula. My university was the first institution to become part of the program and the first to receive their donation. In my class, I used their web applications testing software, their web load testing software, and their product that manages requirements, holds test plans, and tracks defects.

Other companies have donated software (and hardware) to my university for us to use in our classes. The point is that many companies are kind enough to donate their products to colleges and universities for classroom use, but may not do so unless we ask.

Challenges

It was an enormous task, and it was just me doing all this. So, if software companies would like to donate software to a university, they must be prepared to provide extensive training and support. Otherwise, the program will probably fail. In universities where there are multiple faculty involved, it would improve the chances of success, but this will not always be possible. It might also be attractive to have faculty from other disciplines, for example business information systems and computer science.

To successfully build a course and incorporate the software into the curriculum, it takes a great deal of time. At least in my case, I did this in addition to my normal teaching load. Needless to say, I was very busy. So, what would also greatly help the probability of success, is for those organizations interested in working with their local professors, to provide funding to the university for release time. What this means, is that if the professor is teaching two classes per term, funding could be provided to the university to pay the professor's salary for one course, so that they will be teaching one less course. What I need most, is time. At a number of universities, if the economy is in a downturn, teaching loads (the number of courses a professor teaches per term) increase. So we may be teaching even more than usual, not less. Will this cost industry money? Absolutely. But, what is it worth to you to be able to locate qualified college graduates? Think of all the money industry spends on training, once employees are hired.

Another option is for industry to provide money for faculty positions. This would effectively pay a salary (or salaries) for a year. But this type of situation is not permanent, so if you are trying to attract new faculty, you may not be as successful as if you had permanent funding. Having said that, some faculty like to work in "visiting positions" for a year or two, perhaps taking leave from their university. A permanent option is to establish an Endowed Chair position. This position would be funded by a company, and is one of the more expensive options, but perhaps more successful. In such a case, an organization would donate an enormous amount of money, and the interest on that amount funds the position, on an annual basis.

If your university has a Master's Degree or Ph.D. program, then graduate students might be able to work with the faculty members on projects. These projects could, in turn, become a masters or doctoral thesis.

Conclusion

Clearly, both industry and academia can benefit by working together more closely.

What better way is there to get to know the faculty at your local colleges and universities? What better way is there to forge alliances between industry and academia? What better way is there to help ensure that you can recruit students that may meet your needs? There are probably a lot of professors that would like to be "up for adoption"!

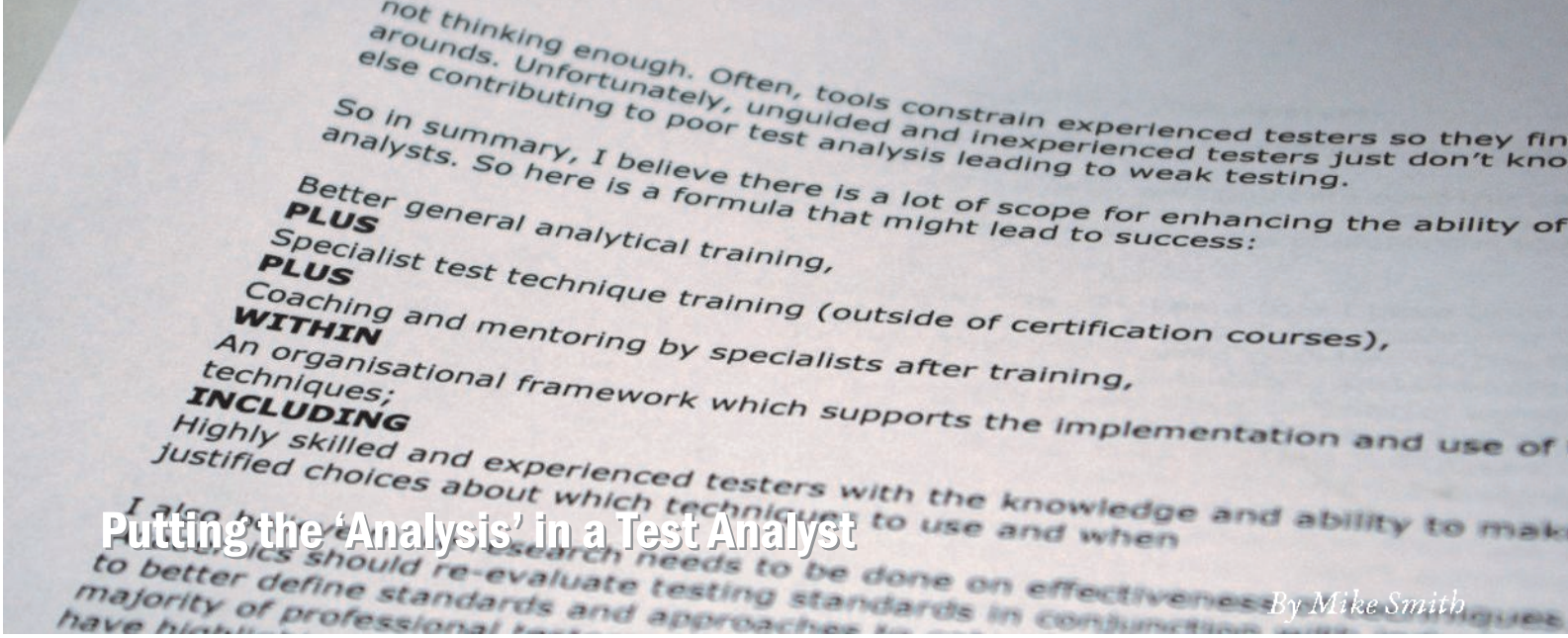
Biography

Patricia A. McQuaid, Ph.D., is a Professor of Information Systems at California Polytechnic State University, USA. She has taught in both the Colleges of Business and Engineering throughout her career and has worked in industry in the banking and manufacturing industries. Her research interests include software testing, software project management, software quality, and software process improvement.

She is the co-founder and President of the American Software Testing Qualifications Board (ASTQB). She has been the person in charge for the Americas, for the Second and Third World Congresses for Software Quality, held in Japan in 2000 and Germany in 2005. She is the Associate Director for the upcoming Fourth World Congress for Software Quality, to be held in the Washington, D.C. area in September, 2008.

She has a doctorate in Computer Science and Engineering, a masters degree in Business, an undergraduate degree in Accounting, and is a Certified Information Systems Auditor (CISA). She is a Certified Tester – Foundation Level (CTFL), through the ISTQB, the International Software Testing Qualifications Board.

Patricia is a member of IEEE, and a Senior Member of the American Society for Quality (ASQ). She is an Associate Editor for the Software Quality Professional journal, and also participates on ASQ's Software Division Council. She was a contributing author to both Volumes I and II of the Fundamental Concepts for the Software Quality Engineer (ASQ Quality Press) books, and is one of the authors of the forthcoming ASQ Software Quality Engineering Handbook (ASQ Quality Press).



Putting the 'Analysis' in a Test Analyst

By Mike Smith

As the software testing profession has grown at an amazing rate over the past 15 years there has been a corresponding increase in all manner of things to support testing including textbooks, tools, techniques, certification, training and conferences.

However, one observation I have made over the 30+ years that I have been involved in IT, and 20+ in testing, is the change in the type of people who move into testing, their background and their basic training. Most of the early 'pioneers' of the testing profession migrated from an IT background of Systems Analysis, possibly preceded by Programming, and had usually being formally trained in a number of areas to support their Analyst role. In my own situation, this included training and project roles in Structured Programming, Systems Analysis, Data Analysis & Process Analysis. I think this background put me in a good position to build my testing career, and it now worries me that while there are some very talented people working in testing, the general lack of analytical training means that they are not as effective as they should be. That is not to say that there haven't been advancements in software testing training including those with an emphasis on analysis. There were very few courses around when I started testing and no formal certification schemes.

But, I hear you say, what about Test Techniques? – they are our analytical 'tools'!

This is, of course, true – but I see several issues with this. All too often, even when I visit companies whose testers have had some training in test techniques, I find little or no evidence of their use. So why is this? I believe the reasons are many and varied. They include:

1. Most testers have very little training in them. Often, their first exposure is in the ISTQB Foundation Certificate course, which is little more than an introduction and overview.

2. Most books and courses necessarily focus on relatively simple examples, often in isolation as individual techniques; there seems no integrated approach to the use of test techniques. The real world is far more challenging with ever-growing complexities and challenges.

3. Testers who go on to Advanced/ Practitioner levels of testing certification get the opportunity to study techniques in more detail. However, and it is a BIG however, it is no good testers getting their first real training on the detailed use of test techniques in an Advanced/Practitioner Certification course. Practical project experience is required to get a grasp of the real-life use of test techniques, so really, separate techniques training and practical implementation should be done prior to attempting an advanced certification course. Certification training should then hone skills to syllabus requirements leading to sucessfully passing the exam. The lack of this approach is, I believe, one of the key reasons behind the very moderate pass rate of post Foundation certification courses.

4. Formal test techniques are not the only analytical weapons in a tester's armoury. Experience, informal deconstruction and just 'tabulating stuff' in ways to suit complex functionality, data, threads, risk and non-functional requirements (to name but a few things) is required to cater for the multi-dimensionality of testing.

5. My experience has shown that just like in many other forms of complex learning areas, follow-up coaching and mentoring is necessary for techniques to be successfully applied in the workplace. Just going on a course and trying to employ them in the workplace without any support from experienced technique practitioners usually fails. This leads neatly on to the last 2 major issues.

6. Many organisations are simply not geared up to be able to effectively implement the use of test techniques. I have seen many testers who achieve Advanced/Practitioner level certification go back to their workplaces only to find practices and processes not in place to support their implementation.

7. Above all, it is the ability & experience required to make & justify the choices in the use of test techniques at both strategic and tactical testing levels that is vital in being able to address the successful implementation of test techniques. Unfortunately, there is very little in the public domain to help. It is expensive to spend time on formal test techniques, so the ability to make appropriate choices backed by metrics including return on investment and risk, is essential to justify the use of them. Senior management buy-in is required to address point 6 above which in turn will then enable point 5 to be addressed. However, this is not all that is required. I once consulted with a major organisation which had 'bought in' to the idea of using test techniques largely because of the 'safety critical' nature of the organisation. They embarked on some education and then 'blindly' followed them in the workplace. They were then shown some more informal techniques and in the end worked out for themselves the best mix of formal and informal techniques that would work for them. The point is, it requires a high degree of skill and experience to make these choices and the absence of any industry benchmarks and standards in this area contributes to making this ability very rare.

I have focussed on test techniques in this article since it should a key part of the analytical skillset of testers. However, there are

other contributory issues that need to be addressed when considering the analytical abilities of testers.

One of these issues is what I call 'Separating the What from the How'. Much of testing has been built up around standards that focus on Test Cases and 'Test Design work products'. These work products tend to combine the 'What' and the 'How' without distinction, and often end up organised in one hierarchical repository. People tend to refer to 'the activity of Test Analysis & Design'. I would rather view this as separate activities of 'Test Analysis' & 'Test Design' leading to separate work products that may have separate structures for their organisation. This better reflects the complex logical to physical relationships that usually exist between requirements and systems as implemented. I see Analysis as setting the measures & targets of success for testing and Design how those measures and targets will be achieved. Many techniques labelled 'Test Design Techniques' are really analytical techniques which set the 'What' to test, represented by Test Conditions. So in my view, more evaluation of when and how to use test techniques according to the point you are at in the development life-cycle is required.

Then there is the issue of tools available to support test analysis. Some of the limitations in current standards and thinking relating to separating the 'What' from the 'How' are perpetuated in commercially available test repositories. I see many testers becoming 'a slave to a tool', expecting the tool to do the job and hence not thinking enough themselves. Often, tools constrain experienced testers so they find work-arounds. Unfortunately, unguided and inexperienced testers just don't know anything else, contributing to poor test analysis, leading to weak testing.

So in summary, I believe there is a lot of scope for enhancing the ability of test analysts. Here is a formula that might lead to success:

Better general analytical training,

PLUS

Specialist test technique training (outside of certification courses),

PLUS

Coaching and mentoring by specialists after training,

WITHIN

An organisational framework which supports the implementation and use of test techniques;

INCLUDING

Highly skilled and experienced testers with the knowledge and ability to make justified choices about which techniques to use and when

I also believe more research needs to be done on effectiveness of techniques. Academics should re-evaluate testing standards in conjunction with industry experts to better define standards and approaches to cater for the challenges faced by the majority of professional testers and should especially look at the 'What'/'How' issue I have highlighted in this article.

Tools and certification can then be improved to better support the real needs of testers. I see the growth in both as very positive but there is much more still to be done to provide a full career development path for testers – and this should include non-testing related education, certification and coaching as well as testing related. Testing is one IT discipline that cuts across all the others and it is vital that testers don't become insular but reach out to all the others. This will not only help them become more effective, but also help those outside of testing who currently don't understand the proper value that it can bring to a project. That way, 'Testing as a profession' can become a reality.

So there is a long way to go. Those that think testing is simple and mostly understood and well-defined need to change their mindset!

Biography

Mike is Managing Director of Testing Solutions Group (TSG) and has a broad background in systems development and testing stretching back 30 years.

After starting his career in pharmaceutical research where he continued his education in Chemistry, he moved into IT where he worked in the Telecoms, Pharmaceuticals and Banking sectors.

Mike became an independent consultant in 1984 and initially worked in the development of critical financial applications. He was the original author of the T-Plan Test Process Management tool which was first implemented by The Bank of England in 1989 (see www.t-plan.co.uk) He also founded ImagoQA Ltd which at the time was the largest independent test consultancy in the UK with global operations in the USA and Australia.

Over the past 20 years, Mike has produced a series of papers about Test Process and Test Management and presented numerous seminars in these subjects together with IT Governance and Information Traceability. Mike was on the presenter list at the very first STAR conference in Las Vegas, 1992 and the most recent STARWest conference in Los Angeles 2007. In October 2007 he presented a keynote at the annual Ericsson Test Conference in Sweden.

Mike is using experience of various entrepreneurial start-ups, board-level appointments as well as his testing and QA background to develop ways in which the profile of testing can be raised in a manner that can be better understood by key business and IT stakeholders.

Mike is Secretary of the UK Testing Board (which represents the UK on the International Software Testing Qualifications Board – ISTQB) and the UK representative on the ISTQB Advanced Level Syllabus Working Party.

Mike's interests include cookery, wine, cycling, reading, quizzes, sport in general especially cricket and football. Mike was appointed a member of the main committee at Essex County Cricket Club in 2007 following over 10 years of involvement in sponsorship, benefit years and sub-committee chairmanship.

A Quality Manifesto

By Tom Gilb

The main idea with this paper is to wake up software engineers, and maybe some systems engineers, about quality. The software engineers (sorry, 'softcrafters') seem to think there is only one type of quality (lack of bugs), and only one place where bugs are found (in programs). My main point here is that the quality question is much broader in scope. The only way to get total necessary quality in software, is to treat the problem like a mature systems engineer. That means to recognize all critically interesting types of quality for your system. It means to take an architecture and engineering approach to delivering necessary quality. It means to stop being so computer program-centric, and to realize that even in the software world, there a lot more design domains than programs. And the software world is intimately entwined with the people and hardware world, and cannot simply try to solve their quality problems in splendid isolation. I offer some principles to bring out these points.

A group of my friends spent the Summer of 2007 emailing discussions about a Software Quality Manifesto. I was so unhappy with the result that I decided to write my own. At least I was unhampered by the committee.

'Software Quality' is a Systems Engineering Job.

Quality Manifesto/Declaration

Proposition:

"**Excellent system qualities** are a continuous management and engineering challenge, with no perfect solutions".

Corollary:

"when **management and engineering fail** to execute their quality responsibilities professionally, the quality levels are accidental; and probably unsatisfactory to most stakeholders."

System Quality can be viewed as a set of quantifiable performance attributes, that describe how well a system performs for stakeholders, under defined conditions, and at a given time.

System Stakeholders judge past, present, and future quality levels; in relationship to own their perceived needs/values.

System Engineers can analyze necessary, and desirable, quality levels; and plan, and manage to deliver, a set of those quality levels, within given constraints, and available resources.

Quality Management is responsible for prioritizing the use of resources, to give a satisfactory fit, for the prioritized levels of quality: and for trying to manage the delivery of a set of qualities - that maximize value for cost - to defined stakeholders.

Quality Principles Heuristics for Action: An Overview.

1. **Quality Design**: Ambitious Quality Levels are designed in, not tested in. This applies to work processes and work products.
2. **Software Environment**: "Software" Quality is totally dependent on its resident system quality, and does not exist alone; 'software qualities' are dependent on a defined system's qualities – including stakeholder perceptions and values.
3. **Quality Entropy**: Existing or planned quality levels will deteriorate in time, under the pressure of other prioritized requirements, and through lack of persistent attention.
4. **Quality Management**: Quality levels can be systematically managed to support a given quality policy. Example : "Value for money first", or "Most competitive World Class Quality Levels".
5. **Quality Engineering**: A set of quality levels can be technically engineered, to meet stakeholder ambitions, within defined constraints, and priorities.
6. **Quality Perception**: Quality is in the eyes of the beholder: objective system quality levels may be simultaneously valued as great for some stakeholders, and terrible for others.
7. **Design Impact on Quality**: any system design component, whatever its intent, will likely have unpredictable main effects, and side effects, on many other quality levels, many constraints, and many resources.
8. **Real Design Impacts**: you cannot be sure of the totality of effects, of a design for quality, on a system, except by measuring them in practice; and even then, you cannot be sure the measure is general, or will persist.
9. **Design Independence**: Quality levels can be measured, and specified, independently of the means (or designs) needed to achieve them
10. **Complex Qualities**: many qualities are best defined as a subjective, but useful, set of elementary quality dimensions; this depends on the degree of control you want over the separate quality dimensions.[1]

Quality Principles Heuristics for Action: detailed remarks

1. **Quality Design**: Ambitious Quality Levels are designed in, not tested in. This applies to work processes and work products.

There is far too much emphasis on testing and reviews, as a means to deal with defects and bugs. It is a well-known paradigm that you 'do not test quality into a system, you design it in'. We can look at this problem from both an economic and an effectiveness point of view.

From an economic point of view, it pays off, by one or two orders of magnitude, to solve problems early. 44%-64% of all coding defects are the results of defects in specifications (requirements, design) given to programmers [Inspection for Managers [ATT, TRW], as reference for this and other facts about test and reviews]. The cost of removal of defects at late stages explodes by 10x to 100x and more. A stitch in time saves nine.

From an effectiveness point of view, both tests and reviews are ineffective. The range of effectiveness is roughly 25% to 75% (probability of actually detecting defects that are present. [Insp. For Mgt., Capers Jones]. Jones reckons that if we had an effective series of about 11 reviews and tests, we could only remove a maximum of 95% of the injected defects. My conclusion is that 'cleaning up injected defects' is a hopeless cause. There are better options.

The interesting option is that 'an ounce of prevention of worth a pound of cure'. We have to learn to avoid the infection of defects in the first place. It is clear that we can reduce the injection rates by at least 100 to 1. Most requirements documents today (my personal client measurements) contain about 100 major defects per page (300 words). The standard that advanced developers (IBM [Humphrey], NASA) have long since established is a tolerance (process exit level) of less than 1.0 majors/page (IBM : 0.25, NASA : 0.10). This is the primary focus of CMMI Level 5 (Defect Prevention Process [Mays, Robert, IBM]. It takes my clients about 6 months to reduce injection by factor ten, and another 2-3 years by another factor 10. This is obviously more cost-effective than waiting until we can test for defects, or until customers complain.

2. **Software Environment**: "Software" Quality is totally dependent on its resident system quality, and does not exist alone; 'software qualities' are dependent on a defined system's qualities – including stakeholder perceptions and values.

We tend to treat software quality as something inherently resident in the software itself. But all qualities (example Security, Usability, Maintainability, Reliability) are highly dependent on people, their qualifications, and they way the use systems. The consequence is that we must plan, specify and design with a stronger eye to identifying and controlling the factors that actually decide the system quality. We have to engineer the system as a whole, not just the 'code'). We must be systems engineers, not program engineers. This has large implications for how we train people, how we organize our work, and how we motivate people. We will also have to shift emphasis from the technology itself (the means) to the results we actually need (the ends, quality requirement levels).

3. **Quality Entropy**: Existing or planned quality levels will deteriorate in time, under the pressure of other prioritized requirements, and through lack of persistent attention.

Even the concept of numeric quality levels, for most qualities – example usability, security, adaptability – is alien to most software engineers, and to far too many systems engineers. But the basic concept of quantified quality levels is old and well established in engineering.

In spite of this poor starting environment, of too many people satisfied with using words ('easy to use') instead of numbers ('30 minutes to learn task X by Employee type Y'), we need to not merely achieve planned quality levels upon initial delivery and acceptance of systems. We need to imbed in the systems the measurement of these qualities, and the warning systems needed to tell us

[1] CE Chapter 5, download, http://www.gilb.com/community/tiki-download_file.php?fileId=26 will give rich illustration to this point. See for example Maintainability, Adaptability and Usability.

they are deteriorating or drastically fallen. We need to expect to take action to improve the quality levels back to planned levels, and perhaps improve them even more in the future.

4. **Quality Management**: Quality levels can be systematically managed to support a given quality policy. Example: "Value for money first", or "Most competitive World Class Quality Levels".

It is useful management if there is a policy about the levels of quality we aspire to, both at a corporate level, and a project level. We cannot really allow isolated individuals to make their dream levels of quality be taken as requirements, without due balance towards the priorities of the other competing levels. And we need to keep our eyes on available resources and technological limits and opportunities.

We need to decide if we are there to 'be the state of the art' (as Rockwell explained to me once) of 'get the most value for money', as others need to worry about.

A policy like this might be generally useful: "Quality levels will be engineered to a level that gives us arguably high return on the investment needed to get them there, and so that the levels do not steal resources for other parallel investment opportunities in quality, or elsewhere.".

5. **Quality Engineering**: A set of quality levels can be technically engineered, to meet stakeholder ambitions, within defined constraints, and priorities.

It is a tricky business to decide which numeric quality levels are appropriate. Initially we cannot decide the right levels in isolation. We need to know about the larger environment, both the environment for the single quality attribute, and for the set of attributes – for their environment.

We need to learn to specify this environment together with the requirement ideas themselves. It will be easier to make decisions about the relative levels of quality and their priority if we have a decisive set of facts about each attribute. For example, it is useful to know things like the:

- Value for a level
- The stakeholders for a quality and for various levels
- The timing needs of levels of quality
- The planned strategies and their expected costs for reaching given levels

And quite a few other things – that will help us reason about the right levels of quality.

Elementary scalar requirement template <with hints>

Tag: <Tag name of the elementary scalar requirement>.
Type:
<{Performance Requirement: {Quality Requirement,
Resource Saving Requirement,
Workload Capacity Requirement},
Resource Requirement: {Financial Requirement,
Time Requirement,
Headcount Requirement,
Others}}>.
========= Basic Information =========
Version: <Date or other version number>.
Status: <{Draft, SQC Exited, Approved, Rejected}>.
Quality Level: <Maximum remaining major defects/page, sample size, date>.
Owner: <Role/e-mail/name of the person responsible for this specification>.
Description: <Optional, full description of the requirement>.
Ambition: <Summarize the ambition level of only the targets below. Give the overall real ambition level in 5-20 words>.
========= Scale of Measure =========
Scale: <Scale of measure for the requirement (States the units of measure for all the targets, constraints and benchmarks) and the scale qualifiers>.
========= Measurement =========
Meter: <The method to be used to obtain measurements on the defined Scale>.
========= Benchmarks ========= "Past Numeric Values" =========
Past [<when, where, if>]: <Past or current level. State if it is an estimate> <-<Source>.
Record [<when, where, if>]: <State-of-the-art level> <- <Source>.
Trend [<when, where, if>]: <Prediction of rate of change or future state-of-the-art level> <- <Source>.
========= Targets ========= "Future Numeric Values" =========
Goal/Budget [<when, where, if>]: <Planned target level> <- <Source>.
Stretch [<when, where, if>]: <Motivating ambition level> <- <Source>.
Wish [<when, where, if>]: <Dream level (unbudgeted)> <- <Source>.
========= Constraints ========= "Specific Restrictions" =========
Fail [<when, where, if>]: <Failure level> <- <Source>.
Survival [<when, where, if>]: <Survival level> <- <Source>.
========= Relationships =========
Is Part Of: <Refer to the tags of any supra-requirements (complex requirements) that this requirement is part of. A hierarchy of tags (For example, A.B.C) is preferable>.
Is Impacted By: <Refer to the tags of any design ideas that impact this requirement> <-<Source>.
Impacts: <Name any requirement or design or plans that are impacted significantly by this>.
========= Priority and Risk Management =========
Rationale: <Justify why this requirement exists>.
Value: <Name (stakeholder, time, place, event): Quantity, or express in words, the value claimed as a result of delivering the requirement>.
Assumptions: <State any assumptions made in connection with this requirement> <-<Source>.
Dependencies: <State anything that achieving the planned requirement level is dependent on> <-<Source>.
Risky: <List or refer to tags of anything that could cause delay or negative impact> <-<Source>.
Priority: <List the tags of any system elements that must be implemented before or after this requirement>.
Issues: <State any known issues>.

Figure 1: an example of a template that tries to collect some of the information that I think we ought to know about in order to decide how to prioritize particular levels of quality for a single attribute. [CE book, page 135]

6. **Quality Perception**: Quality is in the eyes of the beholder: objective system quality levels may be simultaneously valued as great for some stakeholders, and terrible for others.

The point is that any real complex large system will have many different stakeholders. Even one stakeholder category [Novice User, Call Center Manager] can have many individuals, with highly individual needs and priorities. The result will inevitably be a compromise. But we can make that compromise as intelligent as possible. We do not have to design systems with only one level for all stakeholders. We can consciously decide to have different quality levels of the same quality, for different stakeholders, at different times and situations.

For example:

Learnability:

Scale: the time needed for a defined [Stakeholder] to Master a defined [Process].

Goal [Stakeholder = Top Manager, Process = Get Report] 5 minutes.

Goal [Stakeholder = Offshore Clerk, Process = Create New Account] 1 hour.

7. **Design Impact on Quality**: any system design component, whatever its intent, will likely have unpredictable main effects, and side effects, on many other quality levels, many constraints, and many resources.

I see far too much narrow reasoning, of the type: "we are going to achieve great quality X using technology X, Y and Z". This reasoning is not with numbers, but only nice words. Yet I have seen in it $100 million projects, often!

We have to learn to specify, analyze and think in terms of 'multiple numeric impacts of many designs, on our many critical quality and cost requirements'. Quality Function Deployment (QFD) takes this position, but I am not happy with the way in which numbers are used in QFD) – too subjective., too undefined [QFD].

We need to systematically, as best we can, estimate all the multiple effects or each significant design.

8. **Real Design Impacts**: you cannot be sure of the totality of effects, of a design for quality, on a system, except by measuring them in practice; and even then, you cannot be sure the measure is general, or will persist.

I have seen books, papers, and project specifications for software that confidently predict a good result (not usually quantified) from a particular design, solution, architecture or strategy. Maybe it is easier to be confident if no particular numeric impact is ever asserted.

In normal engineering, no matter what the engineering handbook says, no matter what we would like to believe; the prudent engineer takes the trouble to measure the real effects.

We need to carefully do early measurements, then repeat measurements when scaling up, at acceptance times, and later in long-term operation. In we can never take critical qualities for granted, or as if they are stable.

We can plan this in advance to a reasonable degree:

Learnability:

Scale: minutes to learn a Task by a User.

Meter [Weekly Development, 2 Users, 10 Normal tasks]

Meter [Acceptance Test, Duration 60 day, 200 Users, 10 normal tasks, 20 extreme tasks]

Meter [Normal Operation, Sampling Frequency 2%, Tasks = All Defined]

Each 'Meter' specification defines or sketches a different intended test to measure the quality level.

	On-line Support	On-line Help	Picture Handbook	On-line Help + Access Index
Learning Past: 60min. <<-> Plan: 10min.				
Scale Impact	5 min.	10 min.	30 min.	8 min.
Scale Uncertainty	±3min.	±5 min.	±10min.	±5 min.
Percentage Impact	110%	100%	67% (2/3)	104%
Percentage Uncertainty	±6% (3 of 50 minutes)	±10%	±20%?	±10%
Evidence	Project Ajax, 1996, 7 min.	Other Systems	Guess	Other Systems + Guess
Source	Ajax report, p.6	World Report p.17	John B.	World Report p.17 + John B.
Credibility	0.7	0.8	0.2	0.6
Development Cost	120K	25K	10K	26K
Benefit-To-Cost Ratio	110/120 = 0.92	100/25 = 4.0	67/10 = 6.7	104/26 = 4.0
Credibility-adjusted B/C Ratio (to 1 decimal place)	0.92*0.7 = 0.6	4.0*0.8 = 3.2	6.7*0.2 = 1.3	4.0*0.6 = 2.4
Notes: Time Period is two years.	Longer timescale to develop			

Figure 2: A systematic analysis of 4 designs on one quality level (10 minutes). This is an impact estimation table. [CE, page 267].

9. **Design Independence**: Quality levels can be measured, and specified, independently of the means (or designs) needed to achieve them.

There is far too much immediately coupling of named design ideas, with named quality types. "We will improve product agility using structured tools' – type of specification.

We need to focus our specifications on the quality levels we require, and studiously avoid mentioning our favored design idea in the same sentence.

Specifying a 'design', when you need to focus on the quality level, should be considered a major defect in the specification. Dozens or more such 'false requirements' per page of 'requirements' are not uncommon in our 'software' culture.

10. **Complex Qualities**: many qualities are best defined as a subjective, but useful, set of elementary quality dimensions; this depends on the degree of control you want over the separate quality dimensions.[2]

I think there is too little awareness of the fact that quality words often are the name of a set of qualities. The only way to define such complex qualities is to list all the components of the set. Only in this way will we understand what the real requirements are.

We need to learn the general patterns of the most common qualities, as in the example below.

We need to avoid oversimplification of qualities, when, the detailed set of sub-attributes will give us a fair chance at getting control over the critical qualities we want to manage.

Maintainability:
Type: Complex Quality Requirement.
Includes: {Problem Recognition, Administrative Delay, Tool Collection, Problem Analysis, Change Specification, Quality Control, Modification Implementation, Modification Testing {Unit Testing, Integration Testing, Beta Testing, System Testing}, Recovery}.

Problem Recognition:
Scale: Clock hours from defined [Fault Occurrence: Default: Bug occurs in any use or test of system] until fault officially recognized by defined [Recognition Act: Default: Fault is logged electronically].

Administrative Delay:
Scale: Clock hours from defined [Recognition Act] until defined [Correction Action] initiated and assigned to a defined [Maintenance Instance].

Tool Collection:
Scale: Clock hours for defined [Maintenance Instance: Default: Whoever is assigned] to acquire all defined [Tools: Default: all systems and information necessary to analyze, correct and quality control the correction].

Problem Analysis:
Scale: Clock time for the assigned defined [Maintenance Instance] to analyze the fault symptoms and be able to begin to formulate a correction hypothesis.

Change Specification:
Scale: Clock hours needed by defined [Maintenance Instance] to fully and correctly describe the necessary correction actions, according to current applicable standards for this.
Note: This includes any additional time for corrections after quality control and tests.

Quality Control:
Scale: Clock hours for quality control of the correction hypothesis (against relevant standards).

Modification Implementation:
Scale: Clock hours to carry out the correction activity as planned. ''Includes any necessary corrections as a result of quality control or testing.''

Modification Testing:

Unit Testing:
Scale: Clock hours to carry out defined [Unit Test] for the fault correction.

Integration Testing:
Scale: Clock hours to carry out defined [Integration Test] for the fault correction.

Beta Testing:
Scale: Clock hours to carry out defined [Beta Test] for the fault correction before official release of the correction is permitted.

System Testing:
Scale: Clock hours to carry out defined [System Test] for the fault correction.

Recovery:
Scale: Clock hours for defined [User Type] to return system to the state it was in prior to the fault and, to a state ready to continue with work.

Source: The above is an extension of some basic ideas from Ireson, Editor, Reliability Handbook, McGraw Hill, 1966 (Ireson 1966).

Figure 3: An example of Maintainability as a set of other measures of quality. [CE page 156].

[2] CE Chapter 5, download, http://www.gilb.com/community/tiki-download_file.php?fileId=26 will give rich illustration to this point. See for example Maintainability, Adaptability and Usability.

Summary

Purpose [of Quality Manifesto]: To promote a healthy view of software quality.

Gap Analysis:

To help people get to where they really need to be in order to meet their stakeholders expectations as well as resources permit.

Justifications [for positions taken here]

1. We must take a **systems-centric**, not a programming-centric view of quality.

Because: Software only has quality attributes in relation to people, hardware, data, networks, values. It cannot be isolated from the related world that decides

- which quality dimensions are of interest (critical)
- which quality levels are of value to a given set of stakeholders.

2. We must take a **'stakeholder' view** – not customer or user or any much-too-limited limited set of stakeholders.

Because: the qualities that must be engineered and finally present in a software system depend on the entire set of critical stakeholders, not a on a limited few.

3. We must make a clear **distinction** between **various 'defect' types**, as good IEEE engineering standards already do.

Because; we cannot afford to confuse specification defects, with their potential product faults, and product faults with potential product malfunctions. See these definitions.

References

Gilb, Tom, Competitive Engineering [**CE**], A Handbook For Systems Engineering, Requirements Engineering, and Software Engineering Using Planguage, ISBN 0750665076, **2005**, Publisher: Elsevier Butterworth-Heinemann. Sample chapters will be found at Gilb.com.

Chapter 5: Scales of Measure:

http://www.gilb.com/community/tiki-download_file.php?fileId=26

Chapter 10: Evolutionary Project Management:

http://www.gilb.com/community/tiki-download_file.php?fileId=77

Gilb.com: www.gilb.com. our website has a large number of free supporting papers , slides, book manuscripts, case studies and other artifacts which would help the reader go into more depth

For example: Gilb, Inspection for Managers, a set of slides with facts and cases.

http://www.gilb.com/community/tiki-download_file.php?fileId=88

Gilb: What's Wrong with QFD?

http://www.gilb.com/community/tiki-download_file.php?fileId=119

INCOSE Systems Engineering Handbook v. 3

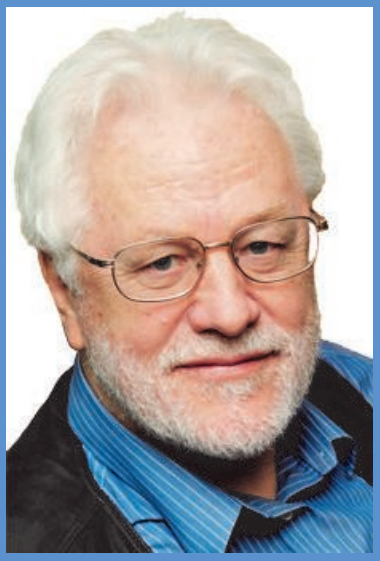
INCOSE-TP-2003-002-03, June 2006 , www.INCOSE.org

Software World Conference Website: Bethesda Md., USA, September 15-18th 2008

http://www.asq509.org/ht/display/EventDetails/i/18370

Source: of Quality Opinions: http://www.qualitydigest.com/html/qualitydef.html [2001]

Version: start 29 oct Monday am 03:00 --à03:36, completed Tuesday AM 00:49 30th oct.



Biography

Tom Gilb is an international consultant, teacher and author.

His 9th book is 'Competitive Engineering: A Handbook For Systems Engineering, Requirements Engineering, and Software Engineering Using Planguage' (August 2005 Publication, Elsevier) which is a definition of the planning language 'Planguage'.

He works with major multinationals such as Credit Suisse, Schlumberger, Bosch, Qualcomm, HP, IBM, Nokia, Ericsson, Motorola, US DOD, UK MOD, Symbian, Philips, Intel, Citigroup, United Health, Boeing, Microsoft, and many smaller and lesser known others See www.Gilb.com .

The Future Tester – What is necessary to know and track?

By Alon Linetzki

When dealing with projects, one has to think people. In the end, we are the weakest link. In the field of high technology, fast-changing environments, and increasing demand for quality at low prices (in short: testers' daily routine), we face a major challenge.

Questions such as "What should we expect from the future test engineer?", "In which issues should we train our teams?, and "How will this support future business and projects?" should be discussed now. Because test managers should be looking at ways to prepare us for these challenges – they are waiting for us just around the corner.

We have all heard about the PTMM, the motivational issues, so we know what WE like to do. What we haven't heard is how industry wants us to be.

When we investigate the needs of the industry (based on a survey of the test engineers market in Israel, which was carried out by SELA Group in the last 2 years), we see a change. From less demanding roles and responsibilities, the tester sought by industry today is required to fulfill a more responsible and technical, yet highly communicative job.

In the past, typical requirements were: UNIX, network, test tools, long hours, independent. Today additional topics have become increasingly significant in the hiring process: team work, cooperative, communicative, independent learning personality, can cope with pressure, knows to identify high risk and low risk, problem-solving skills. These are required in addition to the technical qualifications already in demand.

This creates a dilemma for us testers. For years, the personal issues and personal development have been neglected. We have team leaders that might be the best testers, but maybe they are not so good in their personal skills. We have tight budgets that can hardly fund the technical training, but we would like to use it for the personal issues as well.

Six points relating to 'What should we do'?

1) We should start tracking our employee's knowledge – for a start.

This can be done by simple means, and it only requires an Excel spreadsheet. The data can be used by us for personal development as well as for the technical development of our team.

I'm using the example below for exactly this purpose:

Knowledge items	Appl #1	Appl #2	BP#1	BP#2	UNIX	Total
Dan G.	H	M		L	L	1H, 1M, 2L
Ruth Z.		M	L	M	M	3M, 1L
Total	1H	2M	1L	1M, 1L	1M, 1L	

Appl Legend: H=done design at least once, M=done execution at least once, L=doing execution first time.
Infra legend: H=expert and coach, M=some experience, L=done a course

This matrix helps me to identify the people that learn faster, the people that have little knowledge about something, and the exposure I have in dealing with complex test design and execution. It brings out the best of those people that can learn individually, and allows them to have further topics of knowledge to study. They occasionally have to present what they do, and this brings more responsibility into their work, as well as more satisfaction.

I can also identify problems using the matrix: personal knowledge gaps (i.e. Ruth is no expert in any topic), and project gaps of abilities (i.e. noone has done test design more than once on Appl#2).

2) We should ourselves go to a management course, and start using most of our time for managing people.

If we make a survey to establish how many team leaders and test managers have actually attended a management course (even a general management course), we'll find that only a few have. Large companies have different views on this subject. As I remember from my IBM days, you first have to go to the management course, get the certificate, and only then can you start your role (as test project manager). Other large companies are the same.

The perfect manager uses most if not all of his time for managing others, getting the most out of their abilities and needs. So, if we cannot allocate most of our time to this, we must strive to get things done differently. Maybe we should ask the team what activities they like to do, and try to direct them into doing that? In any case, a simple count of the hours we spend on pure management will tell us right away whether we are on the right track.

3) Attend a skills-oriented course – presentation skills, communication skills, negotiation skills, etc'.

There are many courses on offer that we can pick from. Let's do that. As testers and test leaders we cooperate with many peer groups from day to day, and this can be very beneficial for the day to day communication. I found out great results when my team in one of the projects participated in such a workshop.

4) Link course topics to results and to performance.

Brainstorm this aspect together with your team. How can we link better communication and other skills to performance of our team members and to better results in testing? There can be a lot of parameters to evaluate this, among them: satisfaction of management with the overall performance, satisfaction and relationships with peer groups, better defect ratios (this can be the case!), better fix time from development, better resources utilization (not just people, but the infrastructure people operate).

5) Stay on top of the technical training – do not loose the momentum.

We should try to use the 4th bullet and get the budget for that as well. One option might be that this is accounted to a different budget within the division/department (project budget = technical, division = soft skills).

6) Get interdisciplinary knowledge from other areas of engineering and train your team on it – system engineering, infra, product view, project management, etc'

Having as little as possible on those will boost the project sky-high. Interdisciplinary people are able to analyze things more quickly with a better view on the overall picture. Who else if not we ourselves is in the position to do this? We are system level testers, we have to think systems on a day-to-day basis, and we strive for interaction to get the bugs out.

Summary

On a final note, multidisciplinary people are needed as future testers. People with a lot more emphasis on personal skills than is the case today (not neglecting the technical aspects but enhancing them), and people with interdisciplinary knowledge perspectives on the project in hand.

Biography

Mr. Alon Linetzki, MBA(with excellence), Bs.c., LQA, CSA, ISTQB-CTFL, has been a test engineer and a test manager since 1993, and in IT development since 1983.

His main specialized areas of consulting and training are: * test strategy & risk based testing * test management * test process improvement * defect management & analysis * building & maintaining effective and efficient testing teams.

He is a popular speaker in international testing conferences (STAR, EuroSTAR, Conquest, SQC-ICSTEST, ASTA, Testing & Finance, and more), co-founder of the Israeli Testing Certification Board (2004), and the founder and chair of SIGiST – the Israeli testing forum (2000).

Result-driven testing

By Derk-Jan de Grood

Result-driven testing: Start adding value to your organization

Within the test scene there is a vivid discussion about the necessity and use of a certification model for test experts. The population of testers can be roughly divided into two groups: Firstly, a group that states that it will not do a better job when certified, because the current certifications like ISTQB and ISEB focus on methods and terminology, but fail to look at the practical testing skills of the tester [Bolton, 2007]. Secondly, a group which is pro certification and regards the testing industry as a young yet not fully grown profession that lacks certification models that other professions have been using for ages [Windsor 2007]. Meanwhile, voices are heard that the test profession is accepted in the IT industry and that it has actually grown into a mature profession. The website of the last EuroSTAR conference stated that a mature profession has clearly-defined standards, codes of conduct, and a number of levels of professional competence. Having all those, we might conclude that the testing profession has indeed earned its place among the IT professions.

Have you ever been in a situation where testing time was compromised? Where someone suggested that a lot of time could be gained by expanding the test team with other resources, such as users or administrators that happened to be available? I think most of us have, which shows that there is still some work that needs to be done. The testing community still has difficulty in explaining what the added value of testing is for management, customers and developers [Clermont, 2006]. Many business managers perceive IT as "too little, too late, too costly." There is a gap between business and IT, and on both sides reign incomprehension and a lack of knowledge about the other discipline [Ommeren, 2006]. This is why business management often has wrong expectations of software development and testing. As a result, testing is frequently pushed into a subordinate role and cannot fully contribute to the development process.

Although the discussion about certification is mainly a discussion between testers, it is a rather interesting one. A good certification model will help a tester to prove he masters his profession. A certified tester is knowledgeable on one or more methods, tools and techniques. But that alone will not solve the problems management has with IT. Rather than waiting for the outcome of the discussion about certification, I think action is required today. We need to align the expectations that business management has on IT with the output it delivers. Testing can help bridge the gap since it leads to better software quality and shorter time to market [Grood, 2008]. But this will only happen when the tester is aware of the business needs and acts in conformance with these in order to make the added value of software testing visible in its full colors (see also textbox 1). That's what result-driven testing is about.

Box 1: The added value of testing

- Provided testing is carried out well, it makes a positive contribution to the software development process. Testing leads to better software quality and a shorter time to market by:
- Contributing to the development of "fit-for-purpose" systems that work according to the requirements, quality attributes and expectations (implicit or other). It achieves the goal.
- Preventing damage while the system is live because errors will have been detected during testing and solved on time. This takes away the cause.
- Preventing damage while the system is live because the errors are known and their consequences have been anticipated. This reduces the impact.

- Providing insight into the quality of the test object, which instills confidence in the test object.
- Providing insight into the quality of the test object and its progress, making effective project coordination possible [Black, 2002].

Result-driven testing

Testing is commonly carried out according to a test methodology. The test methodology ensures that testers and other stakeholders use the same terminology and definitions, and that the work is done in a controlled process flow. The methodology also contains best practices so that the knowledge and experience of others can be used. This contributes to the efficiency and quality of the test project. Result-driven testing is more than just applying a methodology. The success of a project is not determined by the methodology, but by the way the methodology is applied. Result-driven testing, therefore, encompasses several aspects:

- A result-driven mindset
- Testing knowledge
- A result-driven approach that supports the application of result-driven testing

A number of good methodologies have already been developed and written about. But like any other profession, testing encompasses more than the simple application of a methodology. After all, strict adherence to a specific methodology is no guarantee for success. Success stems from the mindset, enthusiasm, knowledge and skill of the tester. These factors determine whether a methodology is applied successfully and whether testing is result-driven.

A result-driven mindset -Test principles

Since each project is by definition unique, any method being used will need to be tuned to the situation. Which parts of the method will be used to their full extent? Which steps that are described are not relevant for my situation? Which remarks are relevant but are going to be used in an altered form? Applying the method involves making choices.

What's important to understand are the elements that influence those choices. For one thing it is of course the knowledge and experience we have. We are inclined to choose the options that worked the last time. We are often a little reluctant to choose the options that we have little knowledge of. Another aspect that has significant influence on our decisions is our mindset.

Figure: The success of a project is not determined by the methodology, but by the way the methodology is applied.

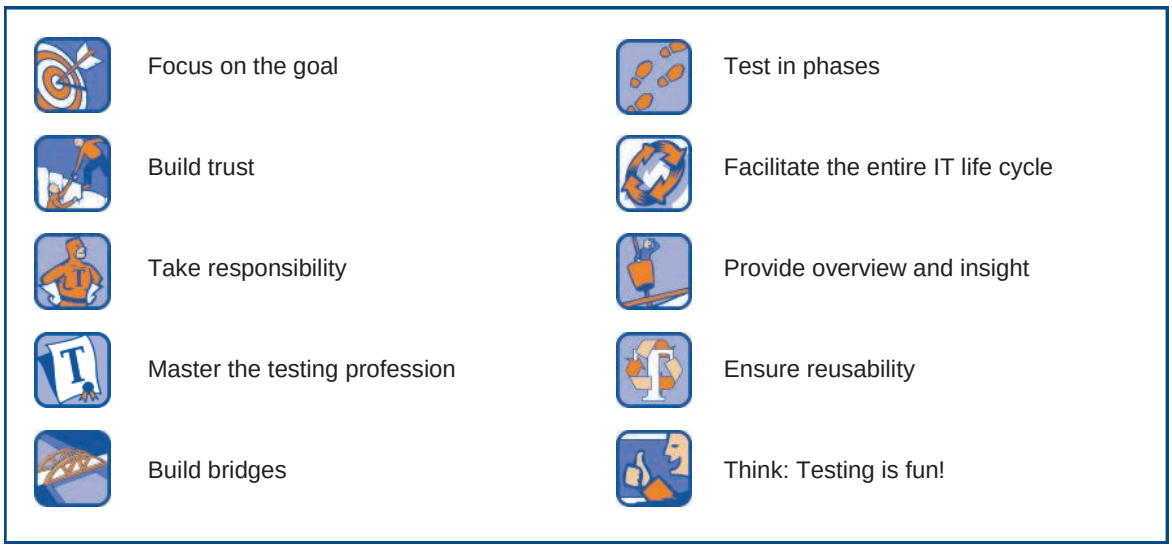
Box 2: The mindset determines the decision

The textbook says: "There are 2 ways of doing this. The first gives the best result, but is complicated. It will take great effort but yields reliable output. The second method is quick and easy to explain. However the results are rough estimates".

Your manager says: "There is little time! We're behind schedule already!" What option do you choose?

The correct application of the methodology depends on the mindset and the expertise of the tester, who has to be able to stress the right things and make the right choices. Ten test principles help result-driven testers to put this into practice. The test principles are a short and strong formulation of the tester's desired mindset, knowledge and working method. They indicate how the tester can create added value and how s/he can make it visible to the organization.

The test principles enable testers to focus on the anticipated goal without external assistance. By thinking and acting according to the ten test principles, testers assume a result-driven mindset. They make sure that they apply the test methodology in such a way that it achieves optimal added value for the organization.



The TestGoal test principles

The ten test principles are not autonomous; they are correlated. Take for example 'Build trust'. This principle aims at creating trust in the product and the test team. Trust in the project will smoothen the go-live decision and is an important motivation for the user to want to work with the new system. This confidence can be created in several ways. One is by showing the organization that certain parts of the system have been realized and are of sufficient quality; that certain risks have been mitigated (provide overview). This leads to a stepwise test approach with distinctive test phases. The go-live decision is also smoothened if all necessary know-how is transferred to the maintenance organization. The principle "Facilitate the entire IT life cycle" has in its origin the understanding that the purpose of a project is not just getting software on the production environment, but support doing business. Building bridges to the ICT Operations department is part of that.

Another way of building trust is providing transparency. By giving insight into the work approach, the stakeholders can judge for themselves how well the team masters the test profession and if it takes its responsibilities serious. This insight is crucial. Otherwise what's the value of a given release advice to stakeholders if they do not know how it has been established? Insight is also given in the way the activities of test team contribute to the set goals of the business and project. One of them could be the delivery of a regression test set, thus ensuring reusability, and once more building bridges to the maintenance department and facilitating the IT life cycle.

The above description is provided as an example of how test principles work in practice. However, we can add more meaning to them by applying them in practice. The reader is therefore challenged to use these test principles and add to them with their own personal experience.

Applying the test principles

The principles can be used as a checklist or as driving force behind the tester's actions.

In the first way, the user regularly checks whether he has given each of the principles enough attention. This will keep him on his toes and focused on providing added value. Too often, certain aspects are forgotten in the heat of the moment. Provided with this self-check the tester has an instrument to evaluate his work."It was hard work delivering the test scripts in time, but unintentionally we compromised the reusability. Maybe we should put some effort in setting this right, next week?"

Using the test principles as a driving force enables the tester to explain his actions. "Your mind is a black box, it takes skill to explain your testing, and what you are accountable for" [Bach, 2007]. The principles are easily understood by non-testers and help to gain commitment for the testers activities. Imagine the situation were you explain that you plan to deliver a test report by the end of the week. The customer responds a little disappointed. "Err…, I don't like reports that much. Besides we only have limited time, so I prefer you use the time for executing tests. That is why we did hire you in the first place, remember?" Explain why you think trust in the system is a requirement for acceptance. In order to gain this trust you'll provide him with an overview and insight. Insight in the quality of the system and the progress of the test process. The principles help communicate (to the stakeholders) why certain activities are important and how they contribute to the anticipated goal.

Either way, a tester who does not apply the principles is doing neither himself nor his customer a favor. It is therefore necessary to regularly check that all of the testers are paying enough attention to all of the test principles. Together, the test principles are the foundation for result-driven testing.

Wanted: result-driven testers

The fast developments in IT make the daily life of the business manager more challenging day by day. The dependency on IT is ever growing. Software can be found in most consumer goods and is increasing in complexity. Most companies earn their money by supplying IT-related services and products, or use IT systems to run their business. The life cycle of software has sharply declined over the past decades. Due to the rapid succession of technological developments and changing markets, systems are constantly adapted. The business manager is dealing with larger uncertainties and has more difficulties to preserve the overview. There is a

demand for people who can help the business manager with easy digestible but accurate information on the status of the project and the risk involved with the new or adapted systems.

Software testers can provide this information and have the power to choose the mode in which his manager operates. We can stick to our complicated test-things and leave the worries to our manager. Or we can adopt a result-driven attitude. We'll take the responsibility to provide and assist the development process and the manager by providing accurate and understandable information. Result-driven testing will lead to happy managers that show appreciation for the test contribution.

If they were not enough arguments for applying the test principles, here's another:

The figure below shows a recent, random picked job-description for a senior test manager. The description contains many references to the TestGoal test principles. It also shows that as part of the bigger concept, certification can be useful as proof of mastering your test profession. Most of all, it tells us that organizations have a strong need for testers who know how to contribute to the business goals. Result-driven testers are wanted!

Position	Senior Testing Manager
Location	Various locations, including 3-4 days per week directly at customer's office
Key Responsibilities	• Leadership and management of one or more testing teams towards attitude to add value to the client, where testing is more than just IT functionality. • Intense client interaction to manage expectations and scope of testing • Set-up test environment and infrastructure to ensure business functionality rather than pure IT check. This covers planning and ensure availability of technical infrastructure at client's side, quality standards as well as intelligent test case set-up and reporting on quality of the tested systems. • Create collaboration model and exploit synergies across all necessary organizational units necessary to deliver seamless services. Planning and organization of all test activities and for delivering test plan, status reports, end reports and execution of test process • Budget responsibility to ensure profitability as return for added value to the customer. • Transformation of current teams and testing offer from time & material to service

References

[Grood, 2008] Derk-Jan de Grood, TestGoal - Result driven testing, Springer, ISBN 978-3-540-7 882 8-7

[Bach, 2007] John Bach, Tutorial on session based exploratory testing at Starwest 2007

[Black, 2002] Rex Black, EuroStar, 2002

[Bolton, 2007] Michael Bolton, Keynote at EuroStar 2007

[Clermont, 2006] Surviving in a QA-Organisation, Markus Clermont, Dutch testing day, 2006

[Ommeren, 2006] De wereld op zijn kop, Erik van Ommeren, IT Beheer, issue 9, 2006, SDU

[Windsor 2007] Susan Windsor, Keynote at EuroStar 2007.

Biography

Derk-Jan de Grood has worked as a tester since 1997. As an ISTQB/ISEB practitioner certified test manager he advises companies on how to set up their test projects. Industries he is familiar with are among others, banking, transport, logistics and telecom.

He is a regular speaker at national and international conferences. Which include: STARwest 2007 (Anaheim, USA), EuroSTAR 2005 (Munich) and 2007 (Stockholm), SQS conference 2007 (Düsseldorf), QA&Testing (Bilbao, Spain, 2007). Derk-Jan is passionate about his profession and gives lectures at various Dutch universities and provides training sessions for fellow professionals in the trade.

In 2007 Derk-Jan published the Book 'TestGoal result driven testing', the English edition will be availabe in June 2008. Within Collis he is responsible for coaching test experts working at Collis. He uses the principles and result driven test philosophy that are described in the book as a daily practice.

Erik van Veenendaal is a leading international consultant and trainer, and recongnised expert is the area of software testing and inspections. He is the director of Improve Quality Services BV. At EuroStar 1999, 2002 and 2005, he was awarded the best tutorial presentation. In 2007 he received the European Testing Excellence Award. He has been working as a test manager and consultant in software quality for almost 20 years.

He is written a number of books, including “The Testing Practitioner” and "Testing according to TMap".

Erik is also a former part-time senior lecturer at the Eindhoven University of Technology, the vice-president of the International Software Testing Qualifications Board and the vice chair of the TMMi Foundation.

Soft Skills: The Forgotten Link!?

A mature profession

The content of this first column in Testing Experiences is based on a number of recent practical experiences. More and more organizations are talking about professionalizing testing by means of defining job profiles, career paths, certification, etc (and rightly so!). The testing profession has become mature and we are being recognized and respected. Taking a closer look at most job profiles, training plans and courses, they are almost always about topics such test strategy, test planning, test techniques and test automation. Test analysts and test managers therefore often have a lot of knowledge and skills regarding testing methods and techniques.

Exciting projects

Most projects implicitly also ask for something else from a professional tester. Things tend to get “exciting” towards the end of a project; the project is delayed, budgets are overrun, users are not satisfied with the functionality, etc. Good reasons for some “problems” between the project sponsor, the business, development and other stakeholders. The tester is often a more or less independent person somewhere in the middle; sometimes more on the user side (acceptance testing) and sometimes slightly more on the development side (system testing). Especially in this type of situation, the tester needs to be able to communicate incidents in a politically correct way, write a test summary report in which the choice of words is of utmost importance, present the status regarding product quality to management, and participate in politically loaded meetings. All of these are activities where soft skills, e.g. on communication, are highly desirable. Possessing soft skills at the required level is not a trivial thing, but is necessary to survive in the complex project world of most testers.

The test training plan

A thorough training plan needs to address, in addition to the test topics mentioned earlier, issues like consultancy skills, written and oral communication, presentation skills, writing reports, and overall social skills. My personal experience with these so-called ‘soft skills” is that they are highly underestimated in career paths and training plans of testers. I’m referring to training courses of at least three or four days and coaching (training-on-the-job) regarding these issues. Also some attention to soft skills in regular test training courses would help.

The next level for the test profession

Of course one can already evaluate the intrinsic characteristics of a person during the initial job interview. Someone who wants to become a professional tester should in fact already by nature have reasonable communication and social skills. Enhancing the soft skills is a step that needs (to get) a lot more attention. Only then can we fulfill our role in projects in a professional manner and show added value. By possessing these skills, we can use our testing method and techniques knowledge and skills to best effect. In practice, however, I see many testers not performing in the way they should; many would benefit from improved soft skills. Let’s take the test profession to the next level!

Improve your profile!

ISTQB Certified Tester Training

- Worldwide -

German, English, Spanish

www.diazhilterscheid.de/schulung

Díaz Hilterscheid

ISTQB Certification: Why You Need It and How to Get It

By Rex Black

If you're a professional tester, test manager, quality assurance staff member, or programmer responsible for testing your own code, you have already discovered that, far from being trivial and straightforward, testing is hard. There's a lot to know. Unfortunately, most practitioners tend to carry out testing as if it were 1976, not 2006. Common practices lag best practices by around 30 years. ISTQB certification is about raising common practices to the level of best practices.

Suppose you are a tester on a project to develop a new system at your company. Somebody e-mails you some screen prototypes with notes that describe the input and output ranges for each field, the actions taken based on particular inputs, and the possible states associated with key objects managed by the system. Would you know how to start designing tests for such a system? ISTQB certified testers do.

Suppose you are a programmer on the same project. You are using a newly purchased tool to help generate and execute unit tests on your code. It reports the statement, branch, condition, and multicondition decision coverage achieved by the tests, and flags constructs that were not tested. Would you know how to create additional tests for the uncovered constructs? ISTQB certified testers do.

Suppose you are a test manager on this project. After four weeks of testing, the project manager asks you, "Based on testing so far, what are the remaining risks to the quality of the system?" Would you know how to do risk-based test status reporting? ISTQB certified testers do.

ISTQB certified testers know how to do these things and more because they have mastered the topics laid out in one or more of the ISTQB syllabi. They feel confident that they have mastered these topics, and can prove to others hat they have, because they have passed one or more of the ISTQB recognized examinations, rigorously developed to check each examinee's abilities to recall, understand, and apply key testing concepts.

In this article, I will explain how the ISTQB certification program works. You'll become familiar with the Foundation and Advanced syllabi, and you'll know where to find online copies of each so you can learn more. I'll tell you how you can prepare for the exam, laying out options from self-guided self-study to attending courses. I'll discuss the exams and what to expect when taking them.

www.istqb.org

What is the ISTQB Program?

In a nutshell, any tester certification program worth your consideration should confirm, through objective, carefully designed examinations, your professional capabilities. Not only does the ISTQB program do so, it is also practical and real-world focused. We address only concepts that you can apply to your work. We support your career path by providing levels of certification that correspond to your experience and roles. Further, we promote and advance software testing as a profession, not merely an ancillary role on a software development team.

So far, the ISTQB program might sound like other tester certification programs you've heard of. Here are a couple unique characteristics. First, the ISTQB syllabi are developed by working groups composed of worldwide experts in the field of software testing, including practitioners, consultants, trainers, and academics. Thus, at each level of the program, you are guided by the distilled wisdom of over 100 experts. Second, while the ISTQB accredits certain training courses, there is no requirement to take expensive training or to purchase pricy study guides. You are free to take the exam with as much or as little help as you need.

ISTQB® Certified Tester

Who Are These People Anyway?

The ISTQB program, like all software tester certifications, is ultimately a *fiat certification*. That is, the people who create, administer, and run the certification stand behind it, and the quality of the certification—or lack thereof—arises from the knowledge, reputation, and abilities of the people involved. So, you'll want to know the ISTQB.

The International Software Testing Qualification Board (ISTQB) is a fast-growing, highly dynamic, and extremely democratic organization. We are a united assembly of member boards, with each board representing major software and system developing countries and regions. Currently, we have 27 member boards, though we intend to admit two or three new boards in our September 2006 meeting and aim to have close to 40 boards by mid-2007. We are a collegial group, focused on cooperating and sharing what works between the member boards. I am the ISTQB President and Erik van Veenendaal is Vice President.

The ISTQB gets its work done through working groups. We have working groups that develop, maintain, and update the various syllabi and the glossary, as well as defining processes and governance matters. Each member board can participate in the working groups. Across the various member boards, we have over 1,000 person-years of testing experience.

The American Software Testing Qualification Board (ASTQB) is the member board for the United States. Like all member boards, we are composed of recognized experts in the field, and include a mix of practitioners, consultants, trainers, and academics. The current officers are Rex Black, President; Patricia McQuaid, Vice President; and, Wayne Middleton, Treasurer. Taz Daughtrey, Jerry Everett, Joe Gance, Andrew Pollner, and Randy Rice are Directors. In addition, Jon Hagar, is an officer in his role as Technical Advisory Group Chair. He and his half-dozen or so technical advisory group members handle the essential member board functions of creating exam questions and exams and accrediting training courses. Finally, Lois Kostroski is the Managing Director, who arranges for administration of the exams, the Website, and other such details.

What is the Value of ISTQB Certification?

The ISTQB program is run by volunteers. Of course, many of us benefit from our involvement, whether in terms of professional prestige, advancement at work, or business opportunities. However, we participate because we believe in this program and the value it delivers.

For the test professional, programmer, manager, and other examinees, holding an ISTQB certificate demonstrates mastery of the best practices and key concepts in the field. This can help advance your career. In addition, by distinguishing yourself from the mass of lesser-qualified test practitioners, holding one or more ISTQB certificates can create opportunities in a competitive, outsourced, and increasingly commoditized job market. So far, test professionals around the world have earned over 35,000 Foundation and Advanced certificates.

Most people who go through ISTQB training and take ISTQB exams are doing so at the behest and at the expense of their employers. Hundreds of smart, well-managed organizations around the world have learned to value the ISTQB certifications because employing ISTQB certified testers ensures better testing, resulting in better software, reduced risk, and lower costs of poor quality, and delivers test consistency and re-usability.

ISTQB Members

- American Software Testing Qualifications Board
- Arabian Gulf Testing Board
- Australian/New Zealand Testing Board
- Austrian Testing Board
- Bangladesh Testing Board
- Brazil Testing Board
- Canadian Testing Board
- Czech and Slovak Testing Board
- Chinese Testing Board
- Danish Testing Board
- Belgium and Netherlands Qualifications Board
- Finnish Software Testing Board
- French Testing Board
- German Testing Board
- Hungarian Testing Board
- Indian Testing Board
- Israeli Testing Certification Board
- Italian Testing Board
- Japanese Testing Board
- Korean Testing Board
- Latin American Testing Board
- Latvia Testing Board
- Malaysian Software Testing Board
- Nigerian Software Testing Board
- Norwegian Testing Board
- Polish Testing Board
- Portuguese Testing Board
- Russian Testing Qualifications Board
- South East European Testing Board
- Spanish Testing Board
- Swedish Software Testing Board
- Swiss Testing Board
- Turkish Testing Board
- UK Testing Board
- Ukraine Testing Board
- Vietnamese Testing Board

Foundations of Software Testing: ISTQB Certification

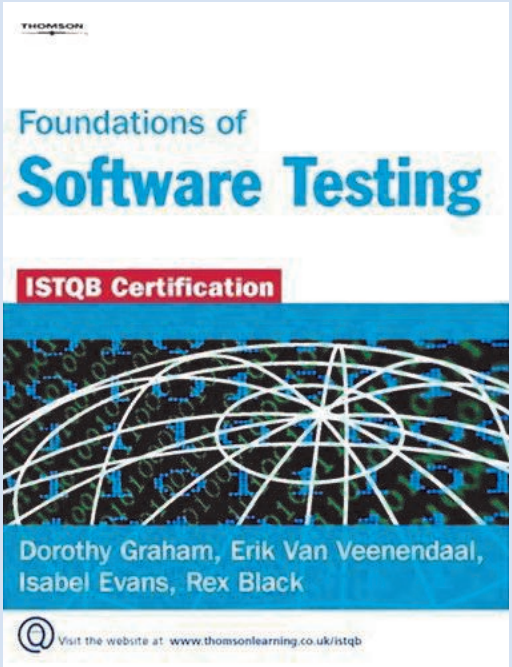
Dot Graham, Erik van Veenendaal, Isabel Evans and Rex Black

Your One-Stop Guide To Passing The ISTQB Foundation Level Exam.

Foundations of Software Testing: ISTQB Certification is your essential guide to software testing and the ISTQB Foundation qualification. This book is an essential purchase if you want to benefit from the knowledge and experience of those involved in the writing of the ISTQB Syllabus.

This book adopts a practical and hands-on approach, covering the fundamental principles that every system and software tester should know. The authors are seasoned test-professionals and developers of the ISTQB syllabus itself, so syllabus coverage is thorough and in-depth. This book is designed to help you pass the ISTQB exam and qualify at Foundation Level, and is enhanced with many useful learning aids. The book also provides sample exam question at the end of each chapter, full-length sample exam paper

Now fully updated for the 2007 Foundation syllabus and new version of the ISTQB Glossary !!



Finally, the ISTQB delivers value to the software testing profession itself. As I mentioned earlier, common practices lag best practices by about 30 years. If programming were stuck in the same rut, 90% of the code would still be written in COBOL, and only 10% of the code would be written in Java. How has software programming progressed while software testing remains stuck in the 70s? Is it that software testers love disco music and sideburns, or has something else happened—or not happened?

Programming, as a profession, has built on its best work. Assembly languages gave way to higher-order languages. Higher-order languages enable structured programming. Structured programming fostered object-oriented programming. Yet, in software testing, we still have practitioners who are unaware of the concepts of equivalence partitioning, boundary value analysis, and error guessing. Worse yet, some people in our field are reinventing and renaming these concepts, leading to inefficiency and confusion.

We at the ISTQB say that we software testers should start building on our best work and stop going in circles. We have learned a lot since Glenford Myers, Richard Bender, Bill Hetzel, and Boris Beizer established software testing as a field. Let's use their concepts and the concepts that have grown from them to define the profession and what professional testers need to know. I invite you to join us as we lead the way towards a truly professional field of software testing, built on a solid foundation.

What Are the Levels of Certification?

The ISTQB certification program is comprised of three levels of certification: Foundation, Advanced, and Expert. At each level, the ISTQB asserts that people claiming to be test professionals should have a particular amount of practical, hands-on experience as well as knowing and being able to apply particular key ideas. The experience required and concepts to be mastered for each level are defined in the *syllabus* (plural, *syllabi*) for that level. Another way to think of the syllabi is as bodies of knowledge or standards for professionalism defined at major stages of a tester's career.

The Foundation Certification is the entry-level certification, designed for people entering the field and for experienced practitioners wishing to start moving up the ladder of ISTQB certifications. The goals of the Foundation Certification are to ensure a broad understanding of the fundamental best practices and key concepts in software testing, and to provide a foundation for professional growth. The syllabus covers six main topics: fundamentals of testing; testing in the software lifecycle; static techniques like reviews; behavioral (black-box) and structural (white-box) test design; test management; and, testing tools. Syllabus-based Foundation training courses are typically three to five days.

The Advanced Certification is the mid-level certification, designed for those with at least five years experience as testers. The goals of the Advanced Certification are to ensure an understanding of advanced best practices and key concepts in software testing amongst committed test professionals, and to support on-going professional growth. The syllabus is divided into three main areas: advanced behavioral (black-box) testing and testing standards for business-oriented testers; test automation and advanced structural (white-box) testing for technically-oriented testers and programmers; and, sophisticated test management concepts for managers. Across all three areas, syllabus-based training courses are typically eight to ten days.

The Expert Certification, which will begin to be deployed in 2007, is for leaders of the field of software testing, those with eight or more years of experience. The goals of the Expert Certification are to ensure consistent understanding and execution of proven cutting-edge techniques by seasoned test professionals, and to lead the software testing profession. Since experts tend to specialize, we will offer various expert syllabi addressing topics like test process improvement, test automation, test management, and industry-specific test techniques. Syllabus-based training courses will again be offered, and, in fact, my company, RBCS, will offer a pilot expert course on test process improvement in 2007.

What Are the Exams Like?

The ISTQB Foundation and Advanced exams contain from 30 to 40 questions. You'll have 60 minutes to complete the exam if you are taking the Foundation exam, and 90 minutes to complete the exam if you are taking an Advanced exam.

There is only one Foundation exam. There are three Advanced exams:

- Functional Tester, targeted mostly at testers doing business-oriented, behavioral (black-box) testing, especially those in independent test teams;
- Test Manager, targeted mostly at test managers, development managers, and project managers responsible for testing;
- Technical Tester, targeted mostly at testers involved in automation, and at testers and programmers who are doing structural (white box) testing.

Each exam is created according to ISTQB guidelines by drawing from a pool of 150 or more carefully crafted questions. Thus, if you retake an exam, you will see different questions.

The exam questions are multiple-choice. However, they are not the relatively easy multiple-choice questions you might remember from high school. These questions are specifically designed—with the help of psychometricians, professional exam creators—to measure the extent to which the examinee has mastered the key concepts in the syllabi.

The key concepts in each syllabus are defined in terms of learning objectives. Each learning objective has an associated level of mastery associated with it: 1) precise recollection; 2) detailed understanding; or, 3) the ability to thoroughly apply the concept to a real-world problem.

On the next page you'll find an example, drawn from my company's Foundation training materials. The Foundation syllabus says that all testers should be able to write test cases using equivalence partitioning and boundary value analysis. Specifically, there is a level-three learning objective associated with being able to do so. The sidebar shows a sample exam question designed to verify a testers mastery of this concept.

Study the sample question. What do you think the right answer is? Can you explain why? See the end of this article to check your work.

Sample Exam Question

A field accepts an integer input from 1 to 99 representing the quantity of an item to be purchased. Consider the following numbers:

I. 0

II. –7

III. 1

IV. 52

V. 99

VI. 100

VII. 129

Which of the following statements are true?

A. II, IV, and VII are boundary values, while are I, III, V, and VI members of the *invalid—too low*, *valid*, and *invalid—too high* equivalence partitions

B. I, III, V, and VI are boundary values, while II, IV, and VII are members of the *invalid—too low*, *valid*, and *invalid—too high* equivalence partitions

C. All seven values are boundary values and members of one of the three equivalence partitions

D. Only I, III, V, and VI are members of one of the three equivalence partitions

Select the one right choice above.

How Do I Prepare for the Exams?

You have four main options to prepare for the exam: self-guided self-study, use of books, e-learning courses, and instructor-lead courses. I'll present these options in that order, though you should not assume that any one option is necessarily better or worse. You have your own needs in terms of studying and preparation, and your own most effective learning style.

Self-guided self-study would occur when you download the syllabus and glossary from the Internet (see below) and then use those documents to guide your preparation. The use of the syllabus and glossary is essential no matter what mode of preparation you choose, but, for very experienced testers who have carried out testing efforts following standards like IEEE 829 and have read two or three of the books on testing referenced in the syllabus, this alone might be sufficient.

As I mentioned, there are books (and standards) which are referenced by the syllabus.

You could pick up a few of these books, along with copies of the standards, and study them, again using the syllabus and glossary as resources and guides. Alternatively, you could buy just one book that is entirely focused on the syllabus you are interested in. Along with Erik van Veenendaal, Dorothy Graham, and Isabel Evans, I wrote such a book on the Foundation syllabus, *Foundations of Software Testing*. I expect similar books at the Advanced syllabus level shortly.

If you find that you have trouble teaching yourself concepts from books and documents, and need more structure to make things stick, then perhaps an e-learning course is for you. For example, Villanova University offers an accredited e-learning course for the Foundation exam. (Full disclosure: My associates and I helped Villanova create this course.) If you don't have time to attend an instructor-lead course but need the structure of a class, this approach might work.

Finally, if you need the structure of a live course, like interacting with fellow professionals, want networking opportunities, and appreciate the convenience of a single, three to five day session followed by the exam, then perhaps an instructor-lead course is for you. Such courses are available in more than 30 countries, including all the countries covered by the ISTQB member boards plus some additional countries. In North America, major ISTQB training providers include my own company, RBCS, along with SQE, Method Park, and ALPI.

No matter what mode of preparation you choose, make sure you do prepare and prepare thoroughly. As I often say to attendees of my training courses, not everyone who doesn't study will fail the exam, but almost everyone who fails the exam didn't study.

Conclusion

So, you should now have a good idea of how the ISTQB certification program works, including the structure and goals of the board. You know the Foundation and Advanced syllabi, and where they fit into your particular career path. You're ready to pick which ISTQB certifications are right for you and start down the path of study and preparation. In short, you are ready to join the thousands of people around the world who are bringing the best ideas and concepts of testing to their day-to-day work, establishing software testing as a profession. I hope you take the next step and become an ISTQB certified tester, as I am proud to be.

Answer to the Sample Exam Question

The correct answer is B. The rule defining valid and invalid inputs creates three equivalence partitions: *invalid—too low*; *valid*; and, *invalid—too high*. Any integer zero or less is a member of the *invalid—too low* partition. Any integer from 1 to 99 is a member of the *valid* partition. Any integer 100 or greater is a member of the *invalid—too high* partition. 0 is the largest member of the *invalid—too low* partition and lies on the boundary with the *valid* partition. 1 and 99 are the smallest and largest members, respectively, of the *valid* partition, lying on the boundaries with the *invalid—too low* partition and the *invalid—too high* partition, respectively. 100 is the smallest member of the *invalid—too high* partition and lies on the boundary with the *valid* partition.

The article was already published in Software Testing and Performance magazine in USA.

Biography

With a quarter-century of software and systems engineering experience, Rex Black is President and Principal Consultant of RBCS, Inc., a leader in software, hardware, and systems testing, as well as part-owner and Chief Technical Officer of PureTesting, a test outsourcing company based in New Delhi, India. For more than a dozen years, RBCS has served its worldwide clientele with training, assessment, consulting, staff augmentation, outsourcing, test automation, and quality assurance services. RBCS has over 100 clients spanning 25 countries on six continents. His popular first book, *Managing the Testing Process*, has sold about 25,000 copies around the world, including Japanese, Chinese, and Indian releases. His three other books on testing, *Foundations of Software Testing*, *Critical Testing Processes* and *Pragmatic Software Testing* (previously published as *Effective and Efficient Software Testing*), have also sold thousands of copies, including Hebrew, Indian, Japanese and Russian editions. He has written over 25 articles, presented hundreds of papers, workshops, and seminars, and given over a dozen keynote speeches at conferences and events around the world. Rex is the President of both the International Software Testing Qualifications Board and the American Software Testing Qualifications Board.

www.rbcs-us.com

Load & Performance:

Tutorial

To join efforts and reach the Everest peak

speakers:

Yaron Tsubery Director QA & Testing, Comverse

Efi Bruker Director R&D, Comverse

Load & performance belong to the top ten list of most frightening words in software development especially when it comes to the point of sale of your product in an extremely competitive market. One other word that is really frightening is the word … STRESS (doesn't it give you shivers just hearing it?). This tutorial will get you acquainted with the terms of the subject and will contribute to your understanding of the whole desired process stages from sales meetings to design and architecture, ways of implementation, testing aspects and finally presenting results and reports. A part of the process you'll be advised about are ways to become aligned with your customer's expectations - means that you need to know and control the required lingo of the load profession, whether you are a sales person, developer or test engineer (e.g. 'usage & traffic model', 'throughput' etc'). The tutorial will present design and architecture solutions along with special guidelines for code writing and implementation. The load test engineers will understand better what the required information to initiate load & performance is, what to search for and how to improve their testing coverage as well as how & what is to be reported and presented in order to give an added value to those who make the decisions.

This practice is focused on projects of complex systems, delivered to telecommunication companies under restricted rules and stiff exit criteria elements in combination with tense delivery timelines.

target audience: Developers, Development Leaders, Development Managers, Load Test Engineers, Load Test Leaders and Testing Managers.

This tutorial will be presented in English language.
Our special offer for this tutorial is 385 Euro.

More information and registration at
www.isqi.org/nc/seminare/rogramm/?sid=168

Berlin, April 28th, 2008

Registration possible until April 7th, 2008.

Díaz Hilterscheid

iSQI

CONQUEST '08

11th international Conference
on Quality Engineering in Software Technology

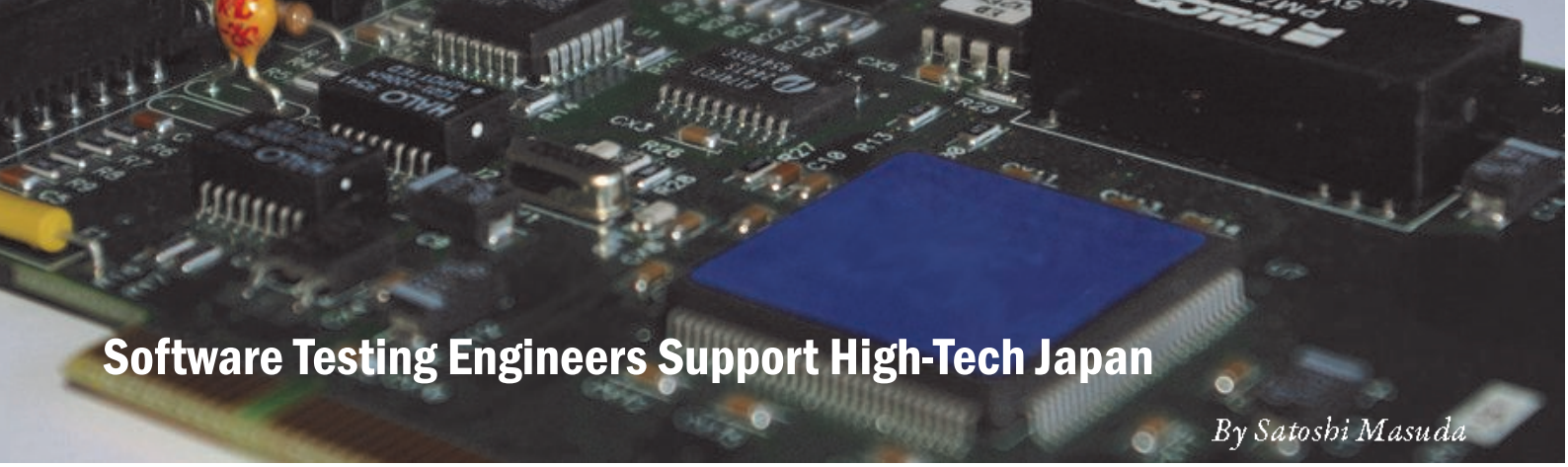
24 - 26 SEPTEMBER 2008 POTSDAM, GERMANY
IHK Potsdam, Breite Straße 2 a-c

Contributions on quality related aspects of software engineering:

- Service Oriented Architecture
- Business Process Engineering
- Secure and Safe Software-Based Systems
- Secure Software Development
- Model-Driven Engineering
- Requirements Engineering
- Verification and Validation
- Testing
- Metrics and Measurements of System Quality and of Development Processes, Analytical Models of Software Engineering
- Project Management
- Configuration Management

Call for Papers: 31 March 2008
Register Now! www.conquest-conference.org

iSQI
International Software Quality Institute



Software Testing Engineers Support High-Tech Japan

By Satoshi Masuda

Japan is famous for high-tech electronics products. Japanese software testing engineers, however, struggle with the bugs in order to support the high-tech products. There are very active communities and study groups for testing engineers. Many books and guidelines have been published which all contribute to software quality. You can learn about testing trends and testing techniques in Japan and about how the Japanese engineers test their software.

Does anybody know about Japanese software testing?

Not many people know how the Japanese engineers test their software. This is due to the fact that the Japanese do not publish their practices very much, or if they do, they often publish in Japanese only. Therefore people in other countries where they do not understand Japanese, may not know about the software testing that is done in Japan.

Japan is well-known in the world for its high-tech and high-quality products, e.g. cars, cameras, videos, games and so on. I guess you, too, enjoy products made in Japan. And maybe you ask yourself: "Do the Japanese make high-quality software the same as their high-quality electronic products?" This article will answer this question.

I will introduce you to what is going on in the Japanese software testing market and will inform you about how Japanese engineers test their software.

Is the quality of Japanese software high?

Regrettably, I must say that the answer is: NO. Japanese engineers, at least, think so, but they think that it is maybe higher than in other countries. The facts are as follows:

Feb. 8, 2008: There was big system trouble in the Tokyo Stock Exchange future market. The market was forced to be close down for half a day. It is said that a program memory initialization error was the cause of this trouble.

Oct, 12, 2007: There was trouble with the automatic ticket gates for the trains service in Tokyo. About 2.6 million (!!) people could not catch their trains in the morning.

Software-related troubles occur frequently and the cause is, in many cases, a software failure. So the quality of Japanese software does not seem to be as high as that of Japanese electronic products. The Japanese think so at least. Therefore, the quality of the software has been focused on very much in recent years. Software testing is a significant part of the quality, and therefore software testing has also been studied very much.

In Japan, embedded software engineers make up a large part of the software industry, mainly because there are so many large companies producing electronics, cars etc.

The Test Engineers' Forum (TEF) was established in 1998 by Dr. Yasuharu Nishi. TEF is the first network for software testing engineers in Japan, so many testing engineers have participated to this Forum. As active members of TEF, they organized the first Japan Symposium on Software Testing (JaSST) [1]. Whilst the attendance at the first symposium was only about one hundred people, the attendance figures have grown steadily every year and reached 1,500 in 2007! The JaSST is now one of the biggest symposiums in Japan. This clearly shows that the Japanese are concerned about software quality and that they do think about how to manage software testing.

The software testing engineers' community in Japan

In this article, I would like to introduce the testing engineers' community in Japan.

As I write this article, there is already a testing engineers' community in Japan, which is organized in the Test Engineers' Forum (TEF). The members are very active and exchange ideas via the Internet mailing list. Unfortunately, this is all done in Japanese only. The topics of the mailing list are very practice-oriented, so that testing engineers can describe their problems via this mailing list. Other engineers can answer them, because they are very conscientious! In this sense, I feel that TEF is not just a community, but is also a place to live together.

Let me tell you more about the achievements of TEF.

TEF translated English software testing books into Japanese. The following are some examples:

- "Testing Computer Software", Cem Kaner,
- "Lessons Learned in Software Testing", Cem Kaner,
- "Managing the Testing Process Second Edtition", Rex Black,

These books have contributed significantly to improving software testing literacy in Japan. Before these books were translated by the TEF, Japanese engineers only had very few software testing books available, e.g.

- "The Art of Software Testing" G.J. Myers
- "Software Testing Techniques" Borris Beizer

These books are without doubt famous and valuable, but they do not provide sufficient details for implementing testing processes and activities in organizations or groups. This is the reason why the TEF translated and published the above books.

Nihon Kagaku Gijyutsu Renmei (Union of Japanese Scientists and Engineers: JUSE) has contributed to the high quality of Japanese products for a long time. JUSE also has a study group for software quality.

In addition to these, there is another software engineers' community, the Software Quality Profession (SQiP) in JUSE. This community includes groups which study software quality from different points of view, e.g. project management, user experience, embedded systems etc. Software testing is one of the SQiP groups. The group studies all aspects of the testing process, test techniques, tools etc. Last December JUSE and Japanese Society

for Quality Control (JSQC) developed and published the Software Quality Body of Knowledge (SQuBok) [2]. These are the first guidelines for software quality knowledge in Japan. JUSE and JSQC spread these guidelines throughout the software testing community and are planning to establish a certification and examination scheme in the future.

Software testing techniques in Japan

Orthogonal arrays and the adaptation of mind -maps to software testing are receiving a lot of attention from software testing engineers in Japan. In the following, I will try to explain the reasons why.

In Japan, focus is given to the adaptation of orthogonal arrays for software testing with a view to reducing the cost of testing. Orthogonal arrays came from Design of Experience (DoE). Genichi Taguchi, a Japanese engineer, researched DoE and developed the "Taguchi Methods". "Taguchi methods are statistical methods developed by Genichi Taguchi to improve the quality of manufactured goods and, more recently, to biotechnology, marketing and advertising. Taguchi methods are considered controversial among some traditional Western statisticians but others accept many of his concepts as being useful additions to the body of knowledge." [3]

By using orthogonal arrays, it is possible to reduce the number of test cases for combination testing, e.g. integration testing, system testing, configuration testing and so on. Japanese engineers are used to orthogonal arrays which are widely used in the industrial sector to test their products. Pairwise testing is similar to orthogonal arrays.

Last year, the book "Software Testing Started on Mind Map" [5] was published in Japan. It is about how to adapt mind maps in order to develop test strategies and design testing architectures. "A mind map is a diagram used to represent words, ideas, tasks or other items linked to and arranged gradually around a central key word or idea." [5]

By using mind maps, a software testing beginner can work as well as an expert in this field. The reasons are as follows:

- The mind map shows how they think about the test strategy and architecture
- It can be described in a MECE sense (Mutually Exclusive and Collectively Exhaustive)
- Experts can follow and review the flow of he beginner's way of thinking.

The book describes why mind maps should be used, and how to use them in software testing. In my opinion, the book is more than that. It will motivate engineers. It shows them a new way of testing, which most people want to try. This book is innovative for software testing.

In summary, it is these two movements of software testing, the traditional way and the new way, which support Japanese high-tech and high-quality products.

Japanese software testing engineers' roles

In the following, I would like to give an insight into the software testing engineers' roles in Japan. You may think the roles are the same the world over. There are, however, subtle differences between Japan and the rest of the world.

First of all, independent software testing has not been recognized throughout the industry. In many companies or organizations, developers have always tested their own software. Some people feel that they design and develop software in high quality, and therefore it is not necessary to test this software. Even though this idea may hold true sometimes, software must still be tested, because software has bugs.

Japanese software testing engineers have to work and face this kind of thinking which sees testing is part of quality assurance. There are so many roles in the software life cycle. Requirements review, for example, is one of the quality assurance roles.

There are some very interesting questionnaires for Japanese software testing engineers. One of the questions reads: "Which animals are similar to the characteristics of a software testing engineer?"

The top answer is a dog! Because they are faithful and hard-working! Japanese testing engineers, too, work faithfully and hard. I think there is a lot of truth in this.

Japanese software testing engineers play certain roles as part of the development team in order to test efficiently and effectively. This is the main role of Japanese software test engineer. Their other role is to be part of quality assurance in order to be involved in the decisions regarding the release of the software. Is the software quality sufficiently high for releasing? How high is "sufficiently high"? This role can be tricky because it may be difficult to answer these kinds of key questions.

Biography

Satoshi Masuda
Director,
Association of Software Testing Engineers (ASTER)
Satoshi Masuda has been involved in Information Technology for 17 years. During this time he has worked for application development and software engineering in a IT leading company. He participated to found NPO Association of Software Testing Engineers (ASTER) in Japan 2005. He is working for establishing Asian Software Testing Alliance among Asian countries.

Conclusion

In this article, I have described the Japanese software testing community, testing engineers and the testing techniques which support Japanese high-tech products. Japanese software sometimes has defects which impact on its society. Many Japanese engineers, however, work conscientiously for better software quality day and night. I would appreciate if this could be understood outside of Japan, and I hope that the Japanese will contribute to software quality in the world.

References

[1] http://www.jasst.jp

[2] http://www.juse.or.jp/software/squbok.html

[3] http://en.wikipedia.org/wiki/Taguchi_methods

[4] http://en.wikipedia.org/wiki/Orthogonal_array

[5] http://www.amazon.co.jp/dp/4774131318

The Need for Measuring Software Security

By Markus Schumacher & Sebastian Schinzel

Given the complex nature of modern software solutions, software testing is a crucial process step in the development cycle. Best practices in software testing are (more or less) standardized and supported with a variety of different tools. As a result, we see complex applications that can be used efficiently without them failing too often. Furthermore, an experienced software tester is able to measure the quality of a software application and compare the results to other software applications.

So far, so good - but we can tell a different story, when it comes to software security. Annual reports of computer security incidents such as *The Web Hacking Incidents Database* regularly reminds us that cyber criminals keep on exploiting security vulnerabilities and literally get their hands on your business assets processed by your computer systems. Most attacks target to steal sensitive information and attackers mostly achieved this goal by exploiting SQL-Injection vulnerabilities. This type of security vulnerability is known for decades, but still widely present in operational application environments.

The Crux of Security Testing

For quite some time, companies and organizations are performing security testing in order to find security vulnerabilities before the bad guys do. But why is the level of software security still so poor? While software testing focuses on positive tests (example: given a certain input, the software application must compute a defined output), security testing is just the opposite – it mostly relies on negative tests (example: "a user must not be able to access other user's data in the database backend". In order to illicitly access restricted information in a database, attackers often exploit *SQL-Injection* vulnerabilities. These vulnerabilities emerge when a developer appends user input to SQL queries without performing proper input validation and output encoding. As an example, let's assume that the following SQL query statement is build to check authentication credentials against a database of user IDs and passwords.

```
String query = "SELECT fld_userId
FROM tbl_users WHERE "
        + "fld_userId='" + userId
        + "' AND password='"
        + fld_password
        + "'"
```

If `userId` is a variable controlled by the attacker, the attacker is able to logon as any user without having the matching password by issuing the string `1' OR '1'='1` as userId. Let's look at the resulting SQL query

```
SELECT fld_userId FROM tbl_users
WHERE fld_userId='1' OR '1'='1'
AND fld_password=''
```

This SQL query will log on the attacker as the user with user ID 1, no matter if the valid password was issued by the attacker or not. One way for developers to prevent SQL injection vulnerabilities is to *SQL-encode* all user input before appending it to SQL queries. Imagine that a developer translates *SQL-encoding* into "prefix all occurrences of an apostrophe in user input with a back slash". This version of *SQL-encoding* transforms the user input *Marc O'Neil* into *Marc O\'Neil*, thus preventing the above SQL-Injection attack shown above. What if the developer uses the following statement to build a SQL query?

```
String query = "SELECT fld_userId
FROM tbl_users WHERE "
        + "fld_userId=" + sqlEn-
code(userId)
        + " AND fld_password='"
        + password
        + "'"
```

Note that `userId` is SQL-encoded before it is appended to the SQL string. Unfortunately, *userId* is not enclosed by apostrophes because the user's ID is expected to be numeric and not a string. If the user enters the string `1 OR 1=1;--` into the username field, the resulting SQL query looks like this:

```
SELECT userId FROM users WHERE
userId=1 OR 1=1;-- AND
fld_password=''
```

This query will successfully authenticate the attacker as user with ID 1 no matter if a valid password was issued or not. The attacker was able to perform an SQL-injection attack against the application even though the application's developer used *SQL-encoding*.

This is just one of many examples of the hidden pitfalls that developers face (assuming that they know about it at all) in their every day work. The complexity of these vulnerabilities makes it difficult for security testers to estimate the security of the software even after a security assessment. Is the application secure, because the security tester does not find vulnerabilities any more? The answer to this question is crucial for the customer and statements such as "almost secure" or "not that secure" are not sufficient. Finding no security vulnerabilities does not mean that the software is secure. It merely means that this tester did not find any security vulnerabilities. Another security tester with higher or different skill may still find many very critical security vulnerabilities. Thus, to make a more precise statement about the security of an application, we define and measure security metrics.

Measuring Software Security

Software quality metrics are used to measure software quality and to compare the quality of applications among each other. This metric-based measurement can also be used to assess the security of an application. To determine fitting metrics, the Goal/Question/Metric paradigm can be used.

As an example, take the above requirement "*A user must not be able to access other user's data in the database backend*" as a goal. As this goal is too generic, it is split into several sub-goals such as "*Non-validated or non-encoded user input must not be copied into SQL query strings*." or better: "*No user input should be copied into SQL query strings*." The last goal is realistic, because for most database systems developers can use prepared statements, which eliminate SQL-injection attacks.

A metric can be formally defined in terms of one of four functions (*understand, evaluate, control, predict*), the attribute of the entity being measured and the goal for the measurement based on the following metrics objective template:

To [understand|evaluate|control|predict] the [entity attribute] in order to [goal(s)]

As a result, one metric to estimate the probability that a software application contains security vulnerabilities is

"To evaluate the number of possible SQL-injection vulnerabilities in order to assure that a user is not able to access other user's data in the database backend."

Using this metric-based approach a security tester is able to measure the security of an application and present the results in a way the customer understands.

Bibliography

Software Security Metrics – ISQI Certified Professional for Secure Software Engineering

Andrew Jaquith, Security Metrics – Replacing Fear, Uncertainty, and Doubt, 2007

Linda Westfall, 12 Steps to Useful Software Metrics

NIST Special Publication 800-55, Security Metrics Guide for Information Technology Systems, 2003

Bernhard Orth: Einführung in die Theorie des Messens, 1995

ISO 9126 Software Engineering Product Quality

Biography

Markus Schumacher is CEO of Virtual Forge, a leading company for software security for business applications. He was a representative of the Fraunhofer-Institute SIT. He worked as product manager and project lead at SAP. He holds a PhD in computer science and is frequent speaker at international conferences.

Biography

Sebastian Schinzel has been a developer and security consultant for more than five years in various technology domains. He focuses on application security assessments, as well as secure development of business software applications. At Virtual Forge, Sebastian is involved in R&D, security processes, and security assessments of SAP customer applications.

TTCN-3: Towards a common testing methodology

By Raquel Jiménez Garrido

In the expansion of telecoms systems there are key factors such as: the growing complexity of the technology, the reduced time to market for new products; interoperatibility between systems, as well as the increasingly high quality demands. All these factors have contributed to decreasing the reliability of the systems.

Testing activities are very important for systems quality assurance and reliability at the beginning of development phases and during the regression phase. Testing and Test Control Notation 3 (TTCN-3) is becoming a safe bet for the telecom companies replacing other testing languages used in the past.

1. Introduction

TTCN is a standard testing language used in Black-Box testing for the specification and implementation of test cases. The TTCN language allows simulating a state diagram of the system under test (SUT). The SUT is a Black-Box that will be excited with stimuli. The answers to those stimuli are observed and compared with the expected answers. Taking in account the results of this comparison, the SUT behaviour can be successful, getting a test case ***pass***; or unsuccessful, getting a test case ***fail***.

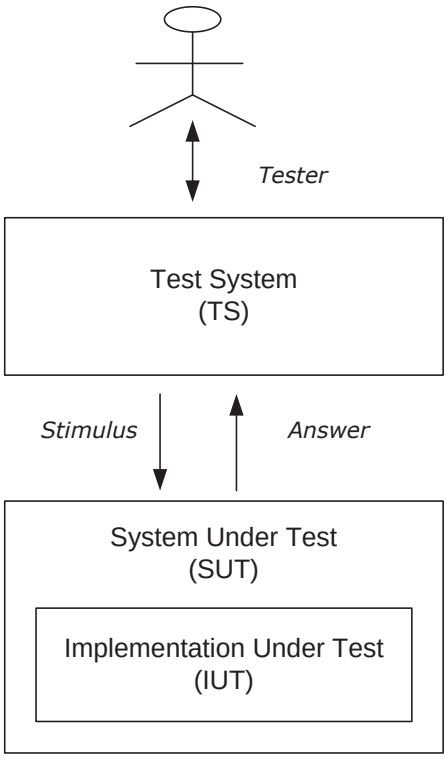


Figure1: Stimulus-Answers Model

1.1 History

TTCN originated within the Conformance Testing framework and is linked to protocol telecommunication testing. "Conformance Testing" is the process verification of a particular vendor implementation. This implementation must be compliant with a determinate standard release. In the first versions, 1 and 2, TTCN behaviour was based on the OSI (Open System Interconnection) representation.

The TTCN-3 evolution has been motivated by several factors such as:

- A language simplification more similar to C language.
- A clear distinction between its homonymous ASN.1 (Abstract Syntax Notation One), which was confusing with TTCN versions 1 and 2.
- New application fields besides telecommunications, such as automotion, avionics or CORBA systems.
- Use in several kinds of testing, such as function testing, load testing, stress testing, interoperability testing; avoiding the use of expensive testing equipment with low flexibility used in that kind of testing.
- Use in complex systems with several nodes e.g. UMTS (Universal Mobile Telecommunications System).

2. TTCN-3 Features

TTCN is a standard language for formal test specification that includes a language description in addition to a mechanism to describe a complete execution environment. The environment description and the core language are shown in Figure 2 and available on the website [10].

TTCN has a set of properties that make it a very flexible language. TTCN has the possibility to import other presentation data into

ETSI	TTCN-3
ES-201 873-1	Part 1: TTCN-3 Core Language
ES-201 873-2	Part 2: Tabular Presentation Format (TFT)
ES-201 873-3	Part 3: Graphical Presentation Format (GFT)
ES-201 873-4	Part 4: Operational Semantics (OS)
ES-201 873-5	Part 5: TTCN-3 Runtime Interface
ES-201 873-6	Part 6: TTCN-3 Control Interface
ES-201 873-7	Part 7: The use of ASN.1
ES-201 873-8	Part 8: The IDL to TTCN-3 Mapping
ES-201 873-10	Part 10: Documentation Comment Specification

Figure 2: Standard specifications

TTCN-3 such as IDL (Interface Definition Language), ASN.1 or XML (Extensible Mark-up Language).

The characteristics listed below make TTCN-3 a testing language which can also be used in other fields than telecom:

- Changing parameter values outside test cases without re-compiling the test cases.
- Automatic checking through matching mechanisms in reception operations
- The default behaviour mechanism to control unexpected events
- Avoiding "ad-hoc" solutions.
- Easy portability and test case re-use
- Allowing to simulate complex systems with different interface communication at the same time
- Code re-use in different testing phases e.g. function test and system test.
- Resource reductions in equipment that is expensive and not very flexible.
- TTCN is independent of a platform or system to test.
- There are already test batteries for technologies such as (Global System for Mobile Communications), TETRA (Trans-European Trunked Radio) or WCDMA (Wideband Code Division Multiple Access).
- Standardization groups such as 3GPP, ETSI, EUROESCOM or ATM Forum have adopted TTCN as a testing language.

The communication mechanism to test the SUT has also been improved compared to previous versions. TTCN-3 allows using two communication ways between different entities involved in the testing process.

TTCN-3 provides an ***asynchronous mechanism communication*** based on messages like in the previous versions. The asynchronous communication is distinguished as a communication between equals: in fact in a client/server scenario the messages are sent and received using the same primitives (***send, receive***) independently of pair role (client/server). The message is sent and the pair continues its behaviour without expecting an answer to the message sent.

Figure 3: Asynchronous communication

TTCN-3 also supplies a ***synchronous communication*** based on procedures. This communication has appeared due to the application of TTCN-3 to new areas. In contrast to message communication, the synchronous communication is characterized by the client/server role and is completely defined in the communication. The client invokes a remote procedure and the server processes this invocation, returning an answer (***reply***) or an error condition which raises an exception (***raise/match***). This communication is done with specific primitives (***call, getcall, reply***…) depending on the pair role. The message is sent and the pair sender is blocked until the receipt of the answer message.

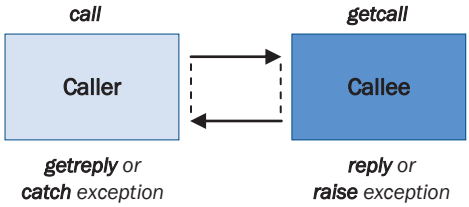


Figure 4: Synchronous communication

Other TTCN-3 features are predefined functions or external functions. The predefined functions facilitate the type language conversion: as ***int2char*** (), which transforms an integer into a charstring or ***oct2bit*** (), which transforms an octetstring into a bitstring. The external functions are developed by the user (e.g. in C++) and can be called inside the TTCN-3 code.

2.1 Test System

Within the test terminology "test suite" is a set or a test battery that has to be passed by the system under test. In TTCN, an "abstract test suite" is used instead; the reason for the use of the word "abstract" is that the TTCN test cases have no information about the system under test.

As a programming language, TTCN-3 is not executable by itself but needs to be interpreted or translated to an executable format. In addition to this, information about the SUT to be executed in needs to be added. The test system parts for the test execution are explained below:

- **System Adapter (SA):** responsible for establishing the communication between the test cases written in TTCN-3 and the real system.
- **Platform Adapter (PA):** responsible for implementing the timers defined in TTCN-3 test cases in the different platforms.
- **Codecs:** responsible for coding/decoding the TTCN-3 messages to the right format to be sent to the SUT.
- **Test Management:** responsible for providing the control part to specify the test cases execution order.
- **Test Logging:** responsible for defining the logs generated during test execution.

3. Application Fields

TTCN was originally used in the telecoms field. TTCN was created to test ISDN (Integrated Service Digital Network) networks and ATM (Asynchronous Transfer Mode); in the 2nd version TTCN-3 was adopted by 3GPP to test mobile handsets. In version 3, it is also used to test Internet protocols. In fact, there are companies dedicated to commercialized test batteries, and even the standardization groups offer test batteries such as SIP (Session Internet Protocol) or H323 protocol.

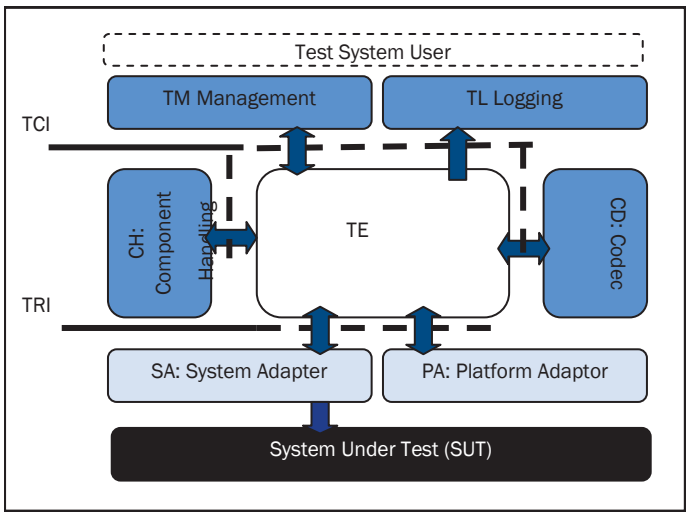


Figure 5: Test System

New application fields, some of them in study, are:

- Web applications; testing hyperlinks, text paragraph, html syntax or format. The study is made at Ottawa University.
- Test Automotive Software, with companies such as DaimlerChrysler or Renault. Using TTCN-3 for testing telematics and electronic systems in vehicles.
- Testing of system trains interlocking in a study by Centrum voor Wiskunde in Informatics, Amsterdam.
- Test components devices over CAN-Bus (e.g. X-ray devices) made by Siemens Medical Solutions.

Biography

Raquel Jiménez has a BS&MSC in Telecommunications and an Executive MBA. She has been working in testing for 7 years, and has been in contact with TTCN from the very start. She has used TTCN applied to test mobile phones and mobile networks, covering technologies such as GSM/GPRS, UMTS or IMS. She is currently working for Métodos y Tecnología as TTCN Services Manager.

Conclusions

TTCN in its version 3 is a versatile testing tool that is increasingly used in many fields also outside telecoms. The use of TTCN-3 by several standardization groups has contributed to its many features, as has the existence of testing batteries, which also increase its popularity as a test language.

TTCN-3 allows having a common testing methodology in the same company. TTCN-3 can be applied to different fields and saves money in human and equipment resources, avoiding "ad-hoc" testing solutions.

TTCN-3 is a tool that can be applied to all testing cycles: from function test, performance test, system test to load test.

As disadvantages, the development of the system adapter and platform adapter could be mentioned. This adaptation, however, is only made once. The statistics say that effort in performing the first test is three times more than doing it manually; but the effort required for subsequent regressions is four times less.

TTCN-3 opens a new market in the testing world; it allows commercializing the "test suites", adaptations (system adapter) or codecs. The test engineer only needs to design the TTCN-3 test cases and the associated parameters.

Cross References

[1] TTCN-3 Standard Part 1: ES 201 873-1 TTCN-3 Core Language.

[2] TTCN-3 Standard Part 2: ES 201 873-2 TTCN-3 Tabular Presentation Format.

[3] TTCN-3 Standard Part 3: ES 201 873-3 TTCN-3 Graphical Presentation Format.

[4] TTCN-3 Standard Part 4: ES 201 873-4 TTCN-3 Operational Semantics.

[5] TTCN-3 Standard Part 5: ES 201 873-5 TTCN-3 Runtime Interface.

[6] TTCN-3 Standard Part 6: ES 201 873-6 TTCN-3 Control Interface.

[7] ITU-T Recommendation X.292: Tree and Tabular Combined Notation.

[8] ISO/IEC 9646-3: Tree and Tabular Combined Notation

[9] TTCN-3 User Conference 2006, Berlin.

[10] Available at: http://www.ttcn-3.org

The human factor within the quality challenge

By Joachim Fuchs

Most companies today accept that quality needs to be inherent throughout the entire application development process. However, recognising the importance of providing code that has been tested for a basic level of functionality and quality requires a cultural shift, for developers, testers and business managers. Developers and testers have traditionally had little in common, sitting on two very different sides of the fence. As such, one approach to encouraging this cultural shift is to get the two to work more closely together by placing testers in the development team to help them run continuous testing cycles throughout the development process. By taking this approach developers will benefit from the wealth of knowledge and experience testers have; likewise testers will understand the many challenges faced by software developers.

As with any cultural change though, much of the impetus must come from the top, otherwise software quality will continue to be a problem for the next 50 years. Developers and testers need to see that quality, not just cost, is a big priority for the company. Business managers can demonstrate this by introducing rewards for the quality of code, not just delivering code on time. In addition, managers cannot expect developers and testers to automatically embrace the change, and understand the role of the other. Few testers or developers have any great knowledge of the way the other works, and most developers will have attended courses that barely touched on the subject of testing. As such, managers must recognise the need for education – giving developers and testers an insight into the role and working practices of the other. Finally, managers must make sure staff are provided with the relevant tools to objectively assess the quality of an application.

Another important change needed to encourage developers to prioritise quality is the introduction of quality gates, or some kind of service level agreement (SLA). A quality gate is a process through which a deliverable must pass before it is accepted by developers as ready for systems testing. Developers need to assess the code they are delivering against quality targets to see if it can pass through the quality gate. If it does not meet the quality targets then the developer needs to take it back for further work, otherwise it can be passed on for systems testing. By implementing this kind of system, businesses are putting in place a formalised process for ensuring the quality of applications, and giving developers the message that they must deliver code written and tested to a basic, pre-agreed standard.

Essentially we are talking about a continuous approach to quality; rather than quality being an afterthought it needs to be a forethought. Quality needs to be written into requirements, and development teams must be given the tools, skills or resources to ensure that testing is integrated into what they do. This isn't something that can happen overnight, but with sponsorship and commitment from the top of the organisation, and changes to the way developers are rewarded, businesses should be able to finally get a handle on the software quality problem.

Biography

Author: Joachim Fuchs, PreSales Consultant Application Delivery Solutions, Compuware

Joachim Fuchs (41) is a certified Computer Scientist and has started his IT career in 1994. Until 2000 he was working in several companies as a consultant for banking software and project manager for Impact Analysis and QA in the Y2k and Euro-conversion area. He has joined Compuware December 2000 as a Presales Consultant for QA Solutions for mainframe and C/S applications. His assignment includes Presales activities as well consultancy and training for the following solution areas: Requirements Management, Test Management, Test Automation.

What a Tester Should Know, At Any Time, Even After Midnight

By Hans Schaefer

You can do the necessary testing «just as a job». This is neither good enough nor is it motivating for the tester. Alternatively, you can invest «the little extra», i.e. do a better job. In this article, I try to define what this «little extra» means.

Major facts are the idea of a tester as a "devil's advocate" in the chase for potential and real defects; the need to prioritize by risk and profit or importance; the use of facts about defects, such as their uneven distribution. But in order to do a good job, the tester must require quality of the product to be tested. There is the Tester's Bill of Rights. Finally there is the need to continuously learn: About defects, the application area, software development, and test methods and tools. This way, testing turns into an interesting and rewarding job, and the tester contributes effectively and efficiently to the project.

1. Testing the Normal Way is not Enough

Systematic testing of software or systems can be learned, just like any engineering discipline. There are tester knowledge certification schemes (ISTQB, ISEB, GTB), there are books (Myers 79, Beizer 95, Kaner 99, Copeland 2004,) and there are standards (ISTQB Glossary, BS 7925, IEEE 829, 1008, 1012,). At least the books and most of the standards have been around for a long time, and many techniques are widely accepted. This means testing can actually be studied and then executed in some systematic way. This does not mean that the typical testing methods described in this material are widely practiced (Schaefer 2004). But it is at least possible to do testing «by following the book».

For a tester, or test engineer, there are two major activities: Designing test cases, and executing test cases and observing and analyzing the results. If the results are not what was expected, deviations must be reported and followed up. Additionally, modern methods, such as exploratory testing (Bach website), include tasks like automation, and management of testing time in the tester's task list.

The normal way of performing a tester's job is to learn some techniques, follow these techniques, execute the test, and conclude the work with a test report. The typical task description tells people to «test the system», wihtout defining any more details. Some books define the task as «getting information about the quality», or «measuring quality» (Hetzel). As test execution is one of the last activities in a project, it is very often run under severe budget and time pressure. This means that the testers have a hidden or open motivation to compromise on the thoroughness of their job. For example, since the detection of defects will always delay both testing and the delivery or acceptance of the product, there is a hidden pressure not to find any defects, or at least not serious ones. Testing is just «done». This job is then not very motivating, investment in learning is scarce, people try to find a different job, and testing does not improve over time.

2. Can We Do Testing in a Better Way?

Glenford Myers, in 1979, defined testing differently: The aim of testing is to find defects. He used this definition because it motivates people. Having a negative focus, trying to break the product, leads to finding more and more serious defects than just checking whether the product «works».

Most people do as they are told. If they are told to find as many defects as possible, they will try to do so. If they are told to get the job done (and explicitly or inherently get the message that defects delay progress), people will try not to find defects, or they will overlook many.

Thus the first rule is to clearly define the purpose of testing, and make the purpose perfectly clear to people. This will be discussed in section 3.

There is an additional problem with any job, not only testing: The world is developing, especially in software. New techniques, methods and tools become available or are used in design. Software products are more and more integrated with each other and growing more and more complex. The focus of the requirements is changing, for example towards emphasizing more security, interoperability and usability. This leads to changes in the requirements on the testing job. Thus, a tester should continuously try to learn more. This will be discussed in section 4.

The next problem is the mindset of people. Some people readily accept information they see or rules that are given. Other people are critical and investigate and ask. As one of the purposes of testing is to find problems and defects, a mindset that does not accept everything without asking, checking more details, getting more information or thinking would lead to better testing. This will be discussed in section 5.

Part of the tester's task is to report incidents. This is not easy. Most literature read by testers just describes issue and defect management, i.e. the life cycle of reporting, prioritizing and handling these. But this is just «the ordinary job». Actually, there is more to it. It can be compared to the task a frustrated user has when calling the supplier help desk: Describing the problem in such a way that the other side accepts it as important enough to do something about it. It means collecting more information about the trouble, but also to think about how to «sell» the bug report to the responsible person. This is the topic of section 6.

Finally, a tester has some rights. We should not just test anything thrown at us over the wall. If information we need is not available, or if the product to be tested is far too bad, testing it anyway would mean to waste resources. There should be some entry criteria to testing, some «Tester's Bill of Rights» (Gilb 2003). This is discussed further in section 7.

All of this has to do with the philosophy of testing. But there are some tools, some very basic rules for doing the work. There is a lot of controversy about what the basis is, but I dare to include a few from a conference speech (Schaefer 2004). This is topic of section 8.

There is definitely more to it. A tester should always be on the outlook for more challenges in field of testing. This paper is only the beginning of what you may look for.

3. The Purpose of Testing

There are a lot of possible goals for testing. One main, though possibly boring, purpose is to measure the quality of a product. Testing is then considered just a usual measuring device. Just usual measuring is not much fun, but is a necessary job, which must be done well. However, there are questions a tester should ask in order to measure optimally. The main question is which quality characteristics are most important to know, and how precisely the measurement must be performed.

Another definition of testing is trying to create confidence in the product. Confidence, however, is best built by trying to destroy it (and not succeeding in doing so). It is like scientific work: Someone proposes a new theory, and all the educated specialists in the area try all they can to show that it is wrong. After this has been tried unsuccessfully for some time, the new theory is slowly adopted. This view supports Myers' definition of software testing: Find defects! The approach is a pessimist's approach. The pessimist believes «this probably does not work» and tries to show it. Every defect found is then a success.

People function by motivation. The definition of testing as actively searching for bugs is motivating, and people find more bugs this way. It works in two ways: One is by designing more destructive test cases or just simply more test cases. The other is by having a closer look at the results, analyzing details a non-motivated tester would overlook. In the latter case this may mean to analyze results that are not directly visible at the screen, but are deep inside some files, databases, buffers or at other places in the network.

A tester should try to find defects!
Defects may appear in places where you do not see them easily, i.e. not on the screen output!

But it is more than this! "Defects are like social creatures: they tend to clump together." (Dorothy Graham, private communication). It is like mosquitoes: If you see and kill one, do you think this is the last one in the area? Thus we may have a deeper look in areas where we find defects. Testers should have an idea where to look more. They should study quality indicators, and reports about them. Indicators may be the actual defect distribution, lack of internal project communication, complexity, changes, new technology, new tools, new programming languages, new or changed people in the project, people conflicts etc. The trouble is that all these indicators may sometimes point in the wrong direction. The defects found may have been detected at nearly clean places, just because the distribution of test cases has tested these areas especially well. Project communication may look awful; some people who should communicate may be far from each other. But such people might communicate well anyway, in informal or unknown ways, or the interface between the parts they are responsible for may have been defined especially well, nearly "idiot-proof". Complex areas may be full of errors. There is a lot of research showing that different complexity indicators may predict defect distribution. However, there are nearly always anomalies. For example, the most experienced people may work with the most com-

plex parts. Changes introduce defects or may be a symptom for areas, which have not been well analyzed and understood. But changes may also have been especially well thought out and inspected. In some projects, there are «dead dogs» in central areas that nobody wants to wake. These central areas are then badly specified, badly understood and may be completely rotten. But since nobody wants to «wake the sleeping dog» there are no change requests. New technology is a risk, partly because technology itself may lead to new threats, partly because using it is new to the people. People do not know what its challenges are. The same is true about the testers. Little may be known about the boundaries of technology and tools. However, it may work the opposite way: New technology may relieve us of a lot of possible defects, or simply make them impossible.

Finally we may look at the people involved. It is the people who introduce the defects. Research has shown that «good» and «bad» programmers have widely differing productivities and defect rates. However, defects do not only result from programming. It is more difficult to map people to design and specification trouble. But at least one factor nearly always has a negative impact: Turnover. If people take over other people's job, they very often do not get the whole picture, because tacit knowledge is not transferred. This is especially true if there was no overlap between people.

Thus, there are lots of indicators that may lead us to more defects, but we have to use them with care.

Defects clump together: they are social!
Defects often have a common cause!
Try to find common causes, and then follow them!

Where you find defects, dig deeper!

Another definition of testing is «measuring risk». This simply means that testing is a kind of risk mitigation, part of risk management. Testers should have an idea about product risk, as well as risk management. In the worst case, testers should ask questions about product risk, especially if nobody else asks these questions.

A very basic method for this is looking at the usage profile, and at possible damage. A tester should ask which kind of user would do what and how often. How will different users use the system? How will this vary over time? And a very important aspect is not to forget about wrong inputs and misuse. People have finger trouble, and interfacing systems have bugs. Thus there is not only use with correct inputs, but probably also a lot of use with wrong inputs. There are three kinds of tests: The good, the bad and the ugly tests: Good means everything is fine, all inputs are right. Bad means inputs are wrong. The ugly tests is where all hell breaks loose: Someone restarts the server while you do your reservation and at the same time sets the machine date wrong, …

The other risk factor is possible damage. This may be difficult to analyze. A start is to at least ask oneself: "What is the worst thing that can happen if this function, feature or request fails?"

Testing is risk mitigation.
What determines risk?
What happens if some input is wrong?

As a summary, it is best if the tester is a pessimist. (A pessimist is an optimist with experience). If something does not work, it is good news, because nobody will have the defect later. The positive effect will be felt in the long run. Better test forces developers to do better work, it informs management about risks, and it leads to lower cost (for repair). Testers bring bad news, but this is their job. Nobody loves speed checks on the motorway! But speed checks make our roads safer, and we all benefit.

A pessimist is a better tester!

4. Continuous Learning

Continuous learning is required in nearly every job. But for testers it is absolutely essential. In most cases, testing is done somehow systematically, using some black box approach. In addition, test design may follow heuristics. Any black box approach may leave important areas uncovered. Any heuristic approach is incomplete, as it is dependent on personal experience (or on learnt experience from others). And white box testing does not uncover errors of omission. It all comes down to this: If there is some aspect the tester doesn't know, it is not tested. Thus the tester should know as much as possible. But how?

A tester needs programming experience. There are lots of programming bugs, even after unit testing by programmers. (Unit testing is often not done well anyway). The tester should have an idea of what is difficult with the particular programming language used. As an example, loops and their counters are difficult in most cases, resulting in off-by-one errors. If the tester has no idea about these, s/he will not check loops with zero, one, maximum and more than maximum iterations, or will not check which individual object is selected. Then off-by-one errors will only be found by chance.

The tester needs design experience. Much design is about contracts and communication: Which module within which constraints and with which reactions should do which tasks? And where are these modules? How do they communicate and solve conflicts? If the tester has no idea about architectures and their inherent problems, s/he will have trouble planning (integration) tests.

The tester needs domain experience. System testing is about implementing the right solution, doing what the domain requires. Can you test a railway interlocking system? (Eh, what does interlocking mean, by the way...?). At least SOME domain experience should be there, and/or communication with different stakeholders.

The trouble is: This is not enough. Much about testing is getting the right information from the right people. Interview techniques are an interesting topic to learn. You get more information if you know how to interview people!

New systems interface with other systems in more and more intricate ways. And there are more and more unexpected interfaces. As an example, someone may integrate YOUR web service into HIS web site, together with other web services. Or your service works in a completely different way than someone else's, and is thus not attractive any more (or much more attractive than expected). This means: Testing for today's stakeholders may definitely not be enough. There

are totally new ways your system may be used or interfaced, totally new ways in which it may be viewed, and you should try to anticipate at least part of this!

A tester should always try to find new ways of looking at the object under test, new approaches, and new viewpoints.

And finally: We want testers to use the newest approaches and technology. You have to learn them. Read testing books, look for and learn tools, study journals, participate in discussion groups, special interest groups, discuss with your colleagues, and go to testing conferences!

Learn more, about everything!
Programming, architecture, new domains, users, tools, anything!

«I am using three things to pull my equipment: dogs, dogs and dogs.» (Roald Amundsen)
For a tester, the three things are:
Learning, learning and learning.

5. A Critical Mindset

Don't believe anything! A colleague of mine said: «believing, that is the activity we do on Sunday morning in the church. Everything else we should check.»

The trouble is: Believing is easier. It does not need any work. We just believe, because something is like expected, or because something is easier. Think about what is written in your Newspaper. Is it really true? Where were the weapons of mass destruction? Was it really the Jews who were responsible for all this bad stuff? Is watching TV really dangerous for your kids? Is a certain soda really good for you?

The answer is: It is easiest not to ask. And if you question everything, you never get anywhere. Thus in our daily lives, we are accustomed not to ask, to take a lot of things for granted. To believe, or to assume, and NOT TO ASK QUESTIONS!

As a tester, don't assume anything. It may be wrong! Designer, specifiers, and programmers assume a lot. It may be difficult to ask because you may look stupid asking. The other part that could answer may be far away or not easily available. You don't even think there may be another interpretation. Or the other part doesn't know, or you get some sarcastic answer...

Using the pessimist view, you may as well assume that any assumption is probably wrong.

Don't assume! Ask!

There are ways to overcome the trouble that you may seem stupid. Learn how to deal with people, learn how to interview, learn how to be self-confident. (Learning, see the section before). Ask someone else. Read, review, sleep over it, and try to find different interpretations. You may need a lawyer's mindset.

If you don't get an answer, have it on your issues list. But don't just assume something! Don't take things for granted! And especially: Don't believe «the system is probably right». There has been at least one banking system paying wrong interest. Difficult to check, after all... There was a geographical information system sending you around half the world instead of the direct way. There are airline reservation systems not telling you all available flights. Many more examples exist.

If nobody else asks the right question, you might do so!
Think about new possibilities, unknown problems, and the stuff you learn.
Think «out of the box»!

6. Defects

Nobody loves the messenger with bad news!

As a tester, most of what you report is bad news. (In a few cases there may be no bad news to bring, because everything seems to work, but that is a very different story).

The bad news is the bugs, or «issues» to call them in a neutral way. Textbooks handle this area well. There is issue reporting, registration, handling, removal, retesting, regression testing. We know all this. But there is something extra to it, and that is not found in the books very much:

1 - An issue is only interesting for a tester if it is accepted as a defect and repaired.

2 - There are defects, which are the result of running many test cases in a row in some very special order.

The first problem is one of salesmanship and discipline. As a tester, one has a sales job. Nobody is interested in spending any money on repairing defects. They will only be repaired if they are important enough. Thus, as a tester you have to report an issue in such a way that the developer understands that it must be repaired. The damage must look severe, the probability of it occurring must look high, and the issue must be easy to repeat.

Thus the tester should not just write an issue on the issues list. The tester should think: Are there any nearby scenarios that would make this problem worse? Are there more stakeholders interested in this? Is this really the whole problem or is there more to it? It is again «thinking out of the box». But it may also mean to invent and run some more test cases. Cem Kaner has presented some excellent material on this cause (Kaner bugadvoc).

Finally, there are the human issues, about diplomacy, politeness etc. A tester should make sure not to hurt anyone personally when reporting an issue. Someone said, "Diplomacy is the art of asking someone to go to hell in such a way that he will enjoy commencing the journey".

For every issue (or bug), research more about it!
Make sure you report it as a risk, and as the whole risk!
Defect reporting is a sales job!
Be diplomatic when reporting issues!

The second problem is worse: Sometimes we experience failures, and we cannot recreate them. For example, the first time the problem occurs, and the next time it is not there. These issues are called »intermittent bugs». They are especially difficult if they introduce system crashes. Upon restarting the system, any corrupted data in the memory may be deleted, destroying the evidence. In many cases, intermittent bugs are the result of long-term corruption of some resource or memory. One example are memory leaks. Some function in the program does not return unused memory after finishing. But because there is a lot of available memory, this can go on unnoticed for a long time, until the memory is depleted. It is even worse if this does not happen every time, but only in very special situations. But also other resources may be depleted. As an example, the Mars Explorer ceased working after 18 days due to too many files accumulated. (Luckily NASA could download new software). In many real-time embedded systems, the tasks are restarted at certain intervals in order to cancel out possible corruption of resources.

The trouble is that ANY shared resource can be corrupted. It comes down to checking the outputs which are not as easily visible as the screen: Files, memory pointers, buffers, databases, messages to remote systems, registry, anything. It could be anything. It could even be the result of a race condition, which depends on the exact timing of some parallel tasks. It is easy to see the screen output; everything else requires tools or extra actions from the tester. This may be too much work to do all the time. And intermittent bugs normally require a whole sequence of test cases, not just one input and output. Finally, it may be somewhere in the operating system, in the libraries, in something that is not your fault.

However, if intermittent bugs occur, it is a good idea to be able to rerun the same sequence of test cases, maybe even with the same timing, and do more checking than before. James Bach (Bach 2005) has a good guide to investigating such problems:

Analyze even intermittent problems!
Log everything you do in testing!
Log everything you see, and look at more remote details!
Make it possible to rerun your tests, with more logging and analysis tools switched on!

One final problem: You may be wrong yourself. Humans err. Testers are humans. This means you overlook problems; you misunderstand outputs, and some of the problems you think you have found are actually no problem. Be self-critical: Sometimes it is your fault. You should also mistrust your memory. It is restricted. This means it is better to take notes, to log what you did, what the system did, what is installed etc. You can trust notes more than your memory.

You may be wrong – don't trust yourself 100%!
Take notes of what you do!

7. The Tester Has Some Rights

As a tester you have some rights.

Testing is often misused by others to clean up the mess. Instead of thinking beforehand, the defects are built into the system and the testers have to find them. This is a waste of both time and effort. A defect found by testers costs many times the effort, which would have prevented it, if it had been prevented. It also leads to extended time to deliver.

Testers should not be used to clean up, but to measure quality and report risk. It is plainly the wrong job.

A tester is NOT a vacuum cleaner!

The answer to the problem is using entry criteria. This means forcing the party before to do the job reasonably well. There are at least two sources where a tester's Bill of Rights has been published: Lisa Crispin talks about testers in Extreme Programming Projects.

The most important tester rights are these three:

- You have the right to make and update your own estimates (…).
- You have the right to the tools you need to do your job (…).
- You have the right to expect your project team, not just yourself, to be responsible for quality.

Tom Gilb (Gilb 2003) developed this list of testers rights (cited with the consent of the author):

Testers Bill of Rights:

1. Testers have the right to sample their process inputs, and reject poor quality work (no entry).
2. Testers have the right to unambiguous and clear requirements.
3. Testers have the right to test evolutionarily early as the system increments.
4. Testers have the right to integrate their test specifications into the other technical specifications.
5. Testers have the right to be a party to setting the quality levels they will test to.
6. Testers have the right to adequate resources to do their job professionally.
7. Testers have the right to an even workload, and to have a life.
8. Testers have the right to specify the consequences of products that they have not been allowed to test properly.
9. Testers have the right to review any specifications that might impact their work.
10. Testers have the right to focus on testing of agreed quality products, and to send poor work back to the source.

The last one is the real clue:

Testers should send bad work back to the source!

8. The Late Night Tester's Toolbox

How should a tester work? What should a tester always keep in mind when working?

One main trouble is to test “everything”. This is far too much. It can never be achieved. But the tester should have some idea about what is tested and what not, or what is tested more or less. This relates to the **test coverage**.

In brief, there are three very basic concepts of coverage, and they can be applied to any notation, which is a diagram. For example a control flow diagram, a data flow diagram, a state transition diagram, a call graph, system architecture or a use case.

- Basic coverage executes every box.
- The next level of coverage is testing every connection.
- This should be the minimum for testing. If there is more time, the next level is combining things, for example pairs of connections.

A tester must be able to state what coverage a test has achieved!

Next, a test should follow the usage profile. This is difficult, especially in module and subsystem testing. But as a tester, one should at least try to get some idea about how the object under test will be used. If nothing can be said, the test should be directed at any use, testing robustness. This means that especially test cases for wrong inputs are of interest.

Follow the usage profile if possible!
If not this is not possible, test for robustness against wrong input.

One technique is the basis of most black box testing: Equivalence partitioning. It helps to save test effort, and it can be applied to derive other test techniques. As a tester, you should know it, but also be aware that it has caveats: Black box testing may leave out important aspects. You should also be aware that combination testing is of interest. Lee Copeland (Copeland 2004) has published a good introduction.

Equivalence partitioning is a good basic technique!
Remember combination testing!

Finally, there is all the test material. A worst-case scenario is if the tester has to admit that the test cannot be done or has been wrong. A big problem is the test environment, which should be prepared and tested early. Waiting for the test environment to work can kill any testing effort (and everybody else will point fingers!). After that, a defect may not be in the object under test, but in the test data or the output analysis. Be self-critical!

Test the test environment – well before test execution!
Check you test data!

And finally, there is test automation. A software product should be soft, i.e. easy to change. Change, however, is a risk. This means there is a need to test after any change. Retesting and regression testing may help. Running tests by using robots helps regression testing. But test automation is more than that: Tools may read specifications and automatically generate test cases. Tools may automatically create environments. Tools may be used to manage the testing effort and the test material.

Automate testing tasks!
Be aware that there is more automation than using test robots!

9. Selected References

Bach 2005: James Bach. A blog note about possible causes of intermittent bugs: http://blackbox.cs.fit.edu/blog/james/

Beizer 95: Boris Beizer, Black Box Testing, John Wiley, 1995

Better Software Magazine, www.bettersoftware.com. www.stickyminds.com Very practical!

BS7925: British Standard: www.testingstandards.co.uk/bs_7925-1.htm

Copeland 2004: Lee Copeland, A Practitioner's Guide to Software Test Design, Artech House, 2004.

Crispin: Lisa Crispin, Tip House, Testing Extreme Programming, Addison-Wesley, 2002, also http://home.att.net/~lisa.crispin/XPTesterBOR.htm

Gilb 2003: "Testers Rights: What Test should demand from others, and why?"., Keynote at EuroSTAR 2003

GTB: German Testing Board: www.german-testing-board.info The German Testing Board developed an earlier version of the current ISTQB certification.

IEEE Standards: See www.ieee.org

ISEB: Information Systems Examinations Board of British Computer Society. http://www.bcs.org/BCS/Products/Qualifications/ISEB/ has run a certification scheme for software testers since 1999.

ISTQB: www.istqb.org International Software Testing Qualifications Board. Develops and runs an international software tester certification scheme.

ISTQB Glossary: www.istqb.org/fileadmin/media/glossary-current.pdf

Kaner 99: C. Kaner, J. Falk, H. Q. Nguyen, "Testing Computer Software (3rd ed.), John Wiley, 1999.

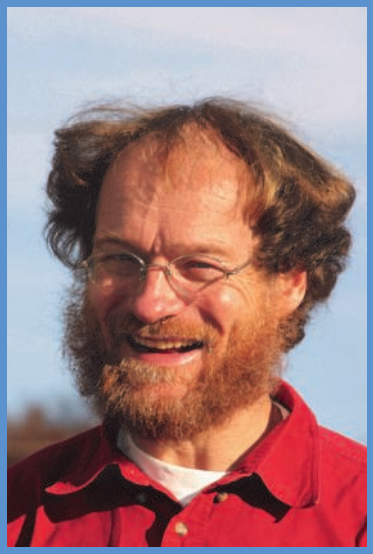
Kaner bugadvoc: A presentation about how a tester should report issues. http://www.kaner.com/pdfs/BugAdvocacy.pdf

Myers 79: Glenford Myers: The Art of Software Testing, John Wiley, 1979.

Schaefer 2004; Hans Schaefer, "What Software People should have known about Software Testing 10 years ago - What they definitely should know today. Why they still don't know it and don't use it", EuroSTAR 2004

About famous software errors: http://wired.com/news/technology/bugs/0,2924,69355,00.html?tw=wn_tophead_1

About assumptions: "Never Assume", Sofie Nordhammen from St. Jude Medical at EuroSTAR 2005. ("Knowing What's Right, Doing What's Wrong")



Biography

54 years old. Specialist in software testing

- 1979 M. Eng.from Technical University of Braunschweig, Germany. Computer science, railway signalling
- 1979 - 1981: Development of real time process control software at the Fraunhofer-Institute in Karlsruhe, Germany.
- 1981 - 1987: Work with Center for Industrial Research in Oslo, Norway. Development of parts of an IDE-tool (Systemator). Later developing test tools and leader of quality improvement program. Consulting and lecturing about software quality.
- 1987 - 2007: Independent consultant in software testing and testing improvement matters. Guest lectures at several universities in Norway about Quality Assurance and Software Testing. Public and in-house seminars in Software Review and Testing in Scandinavian and European countries. Regularly speaking at conferences. Several best paper awards. (Best presentation award at CONQUEST 2003, best paper in 2004).
- 1998 - 1999 Test coordinator in Norwegian Telecom's Y2K project.
- 1999 ISEB Certified Tester (Foundation Level)
- 2004 ISTQB Certified Tester (Foundation and Advanced Levels)
- 2003 - 2007 Chairman of Norwegian Testing Board
- 1992 - 2007 Active participant in running museum trains in Norway, certified for safety critical services on steam locomotives.

a matter of trust!

Testing and Development Services in Latin America

www.bicorp.biz

HOTLINE: +591 3 3397 145

contact@bicorp.biz

We work in English, German and Spanish

Business Innovations S.R.L.

Saavedra Esq. Cochabamba - Torre Empresarial Cainco - piso 13 OF 2

Santa Cruz de la Sierra - Bolivia

Why can't testers code?

By James Lyndsay

Perhaps it's the circles I move in, but it seems that I can't get through a conference without someone telling me, in some fashion: "Coders don't test". If it's not hearing the latest coding-idiot bug, it's the refrain that testers need to be independent - even isolated - from the bug-forgiving influence of the design teams, or someone whining that their coding colleagues aren't remotely interested in all the good work testers put in. I've heard testers gasp in horror on discovering that the coders are writing their own tests, without benefit of ISTQB qualification or the most basic grounding in test design. Something clearly needs to be done.

On the other hand, I've worked in organisations where testing has been an innovation-free zone. Some teams are still writing test scripts in Word, and and printing them out before ticking the boxes. Some teams manually configure their test environment, every time, before laboriously hand-crafting their data. Some check reports by eye, and rely on their memory to remind them about whatever they did just before finding their most recent bug. When I teach exploratory testing, I'm always surprised to find people in teams who hardly use tools, don't share the tools they might use, and certainly don't have the wherewithall even to chuck together a quick script to check that the parameters of the test environment are much the same this morning as they were last night. Curiously, it's these testers who seem most disturbed that coders are writing their own tests.

Here, then, is my take on the matter. It is not that coders don't test, it is that testers can't code. This is a skills gap - whereas coders can test, but sometimes don't, it's uncommon to find a tester who can read code, control a debugger, and fix the bugs they find. Is it any wonder that testers don't get no respect, that testers can't communicate with coders, that testers simply don't understand the range of bugs that using = rather than == can trigger? Is it any wonder that other people on the team are starting to take over the tester's job with vast numbers of unit tests, and business acceptance tests that actually make sense to the business?

Some of you reading this will be quite rightly offended by my blanket characterisation of technology-adverse testers. Some of you may already be excellent coders, whipping up a bug-tracking system or database loading script in your lunchbreak. For the rest of you, perhaps, this will be a call to arms: your colleagues are doing the interesting parts of your job, and you're rolling over to let them. Ignore, for now, the proprietary tools that let you test your whole application with a single button press - don't be tempted by their scriptless attractions. Turn instead to your library, bookshop or web browser, get hold of a beginners guide to a language you can code on your own laptop, and get moving. Write small, and write useful - something to spot changes or load data. Don't try to write automated tests. If you fancy, use a language that your coders use, get them to show you their development environment, the way they build their code and make their living. Perhaps you'd prefer to turn to the powerful and simple unix scripting tools, or their equivalents on any operating system worth a mention. Perhaps you'll pick up Python, Perl or PHP. Any way you choose, engage with the technology that powers our business, get over the hump, and become a tester who codes.

Biography

Based in London, James Lyndsay has been testing since 1986. He formed Workroom Productions in 1994, and has been an independent test strategist ever since. Having started coding in 1981, James has forgotten most of what he once knew, but currently codes rather poorly in combinations of PHP, unix scripts, and ActionScript/Flash. He feels this gives him many powerful test techniques, primarily empathy with the frustrations of professional coders. James was an internal irritant to the ISEB exam process for five years, is a regular speaker and occasional teacher, and runs LEWT (the London Exploratory Workshop in Testing). Visit http://www.workroom-productions.com for papers, recorded talks and tools.

Future public engagements:

UK: London SIGiST, special session on Diagnosis, keynote on "Testing in an Agile Environment" 18 March

Denmark: Exploratory Testing Seminar, 29 April

USA: STAREast Diagnosis tutorial May 6

The long way to become a tester. My experience.

By Tamás Stöckert

I have now worked as a tester for more than five years. My becoming a tester, however, was quite an unusual story: Having worked in Germany and Austria for several years, I decided to move back to Hungary, and so I sent my CV to a headhunter in Budapest.

Within a few days I got a call informing me that people with good communication skills are required by „Lufthansa".

My first thought was: "Hmmm....maybe something to do with passengers?" During the interview it turned out that it was Lufthansa Systems, the IT subsidiary of Lufthansa, that was offering the job.

My first assignment was project coordination in the field of customer management.

After the project was succesfully completed, I was offered a position in the newly founded test team of Lufthansa Systems Hungária on the basis of my background, ...and that is how it all began. Structured, professional testing was something I had already learnt. Nevertheless, I found the area new and exciting, especially since Lufthansa Systems Hungaria was a forerunner in this field. In addition to this, I had the feeling that testing as a whole was gaining significance by the day... Initially, there were only three of us in the core team: Our leader with already significant software testing experience gained in various multinational companies, and the two of us as junior testers with IT know-how, good communication skills and a customer relationship background. A few days later I was already sitting by a customer: my first test assignment had started.

I have to admit that as a junior it was not easy. Questions came to my mind that made me uncertain: Even though my role and my tasks were clear, what is my added value to the project?

The idea of destroying the tiresome work of others, to be "picky" in a good sense, was not really appealing to me at all.

After several months spent in various small projects, I slowly but surely started to realise the high-level objectives of my activity: Find the bugs before the customer finds them, try to find them as early as possible, develop a kind of passion towards quality by appying a structured approach, and last but not least, have fun!

I will never forget the pride I felt when detecting the first critical error at an early stage of a functional testing phase....Yessss! That is what is called success! From that point on, I wanted to learn as much as possible about this area and spread the culture of early failure detection within the company.

Our testing-specific training was pre-planned and professional. Many, many training courses and books (testing, development methodologies, UML etc.) followed coupled with new, bigger and ever more interesting multinational projects.

Only a year after my carreer start, I got my first international certification (ISEB Foundation Level) and with it came the recognition in the company as well...

Slowly I found myself in the position of being a consultant in different development projects, explaining to development leads and project managers the general testing approach and characteristics that should be considered.

In the meantime, our team has grown and the necessity of coordinating the different testing activities turned out to be inevitable. Courses in project management, change management as well as soft skills gave me directions to manage my first assignments in this area.

As a result of a reorganisation in my business unit in 2006, a new test team was formed and I was appointed as the leader of this team. The expectations were clear: Testing competence must be built up by applying a unified training scheme, the team of then 11 colleagues must grow, the culture of testing must be spread within the company.

At this stage, I would like to take the opportunity and briefly present my company: Lufthansa Systems is one of the leading full-service IT providers in the aviation business. Our core business competence is aviation yet LSY has customers from a large variety of other businesses as well. Steady growths and fierce competition on the airline market forces airlines to optimise their business processes and continuously reduce their costs. One means of cost reduction is using fast and reliable IT solutions. Needless to emphasise the vital role of quality.

With the company's headquarters and the majority of the approx. 3500 employees being in Germany and the test team based in Budapest, the vision of an independent test service is not far fetched but rather reality.

A year and a half after my appointment, the results of our hard work became more and more obvious: New, dedicated and highly motivated IT specialists were employed and trained specifically (ISEB Practitioner / ISTQB Advanced Level traning courses), success in various projects, the majority of the team members speak a common testing language. In fact, all team members are ISTQB certified. Right now the team consists of 22 testing professionals

As for me, I passed both the ISEB Practitoner and the ISTQB Advanced Level exams, and as a result of this my testing horizon has widened.

Parallel to the above-mentioned developments in my company, the Hungarian testing community as such has also started to discover itself. Initiatives for establishing the Hungarian Testing Board, as a member organisation of ISTQB, started early last year and by October 2007 all the preconditions of forming the board were fulfilled.

The HTB was founded on 6th October 2007 and on 13th October 2007 its application for ISTQB membership was accepted!

Biography

Tamás Stöckert has been working in different software testing positions for more than five years now. As one of the pioneers in testing by Lufthansa Systems Hungaria he had the possibility to build up significant testing specific know how both from the professional and from the management point of view.

Tamas holds a BSc in economics and an ISTQB foundation as well as ISEB Practitioner / ISTQB full advanced level certificates.

While working for Lufthansa he accomplished a year long management development program organised by the Lufthansa School of Business.

He is fluent in German and English (University of Cambridge Certificate of Proficiency in English) and speaks French as well.

From September 2006 on Tamás leads an ever growing dedicated test team by Lufthansa Systems Hungaria currently counting 22 young yet already experienced and motivated specialists. With his team he is providing complete test service as well as trainings and consultancy mainly for multinational software development projects.

Díaz Hilterscheid

Díaz & Hilterscheid Unternehmensberatung GmbH

Kurfürstendamm 179
10707 Berlin
Germany

Phone: +49 (0)30 74 76 28-0
Fax: +49 (0)30 74 76 28-99
E-Mail: info@diazhilterscheid.de

Editor:
José Díaz

Layout & Design:
Katrin Schülke

Website:
www.testingexperience.com

If you want to write, please contact:
editorial@testingexperience.com

If you want to advertise, please contact:
sales@testingexperience.com

Subscribe:
www.testingexperience.com/subscribe.php

Price:
free of charge

Der SOA-Effekt: beschleunigen Sie Ihre IT

Eine Architektur, die die Unterstützung der Geschäftsprozesse in den Mittelpunkt einer flexiblen IT stellt: Das ist SOA

Lesen Sie, was Serviceorientierte Architektur ausmacht, und wie Sie mit SOA Ihr Unternehmen zu mehr Agilität und höherem Erfolg führen.

Inhalte des Buchs:

- Die Philosophie und wichtigen Basics
- Neue Geschäftsmodelle durch SOA
- SOA-Initiativen planen und durchführen
- SOA-Governance
- Qualitätsmanagement von Services
- SOA nachhaltig implementieren
- Praxisbeispiele aus den Branchen: Automotive, Telecommunication und Banking

Bestellformular, Fax Nr. 02 11/8 66 93 23

Jetzt bestellen und 5,- Euro Download-Gutschein sichern

Ich bestelle ___ Exemplar(e) von
SOA für agile Unternehmen
zum Stückpreis von 39,90 Euro.

Lieferadresse:

_____

_____

_____

_____

_____ _____

Datum Unterschrift

SOA für agile Unternehmen
Serviceorientierte Architekturen verstehen, einführen und nutzen

A. Beinhauer, M. Herr, A. Schmidt (Hrsg.)
Hardcover, 376 Seiten
mit zahlreichen Abbildungen
ISBN 978-3-939707-14-1
Preis 39,90 (incl. MwSt. und Versandkosten)
Symposion Publishing 2008
www.symposion.de

symposion

What about your intellect?

CaseMaker – the tool for test case design and test data generation

If you are ISTQB Certified Tester, you can get a **free version** of the **new** client software for **12 months**.
Please register at www.casemaker.eu/istqbct

CaseMaker®

www.casemaker.eu