June, 2008

testing experience

The Magazine for Professional Testers

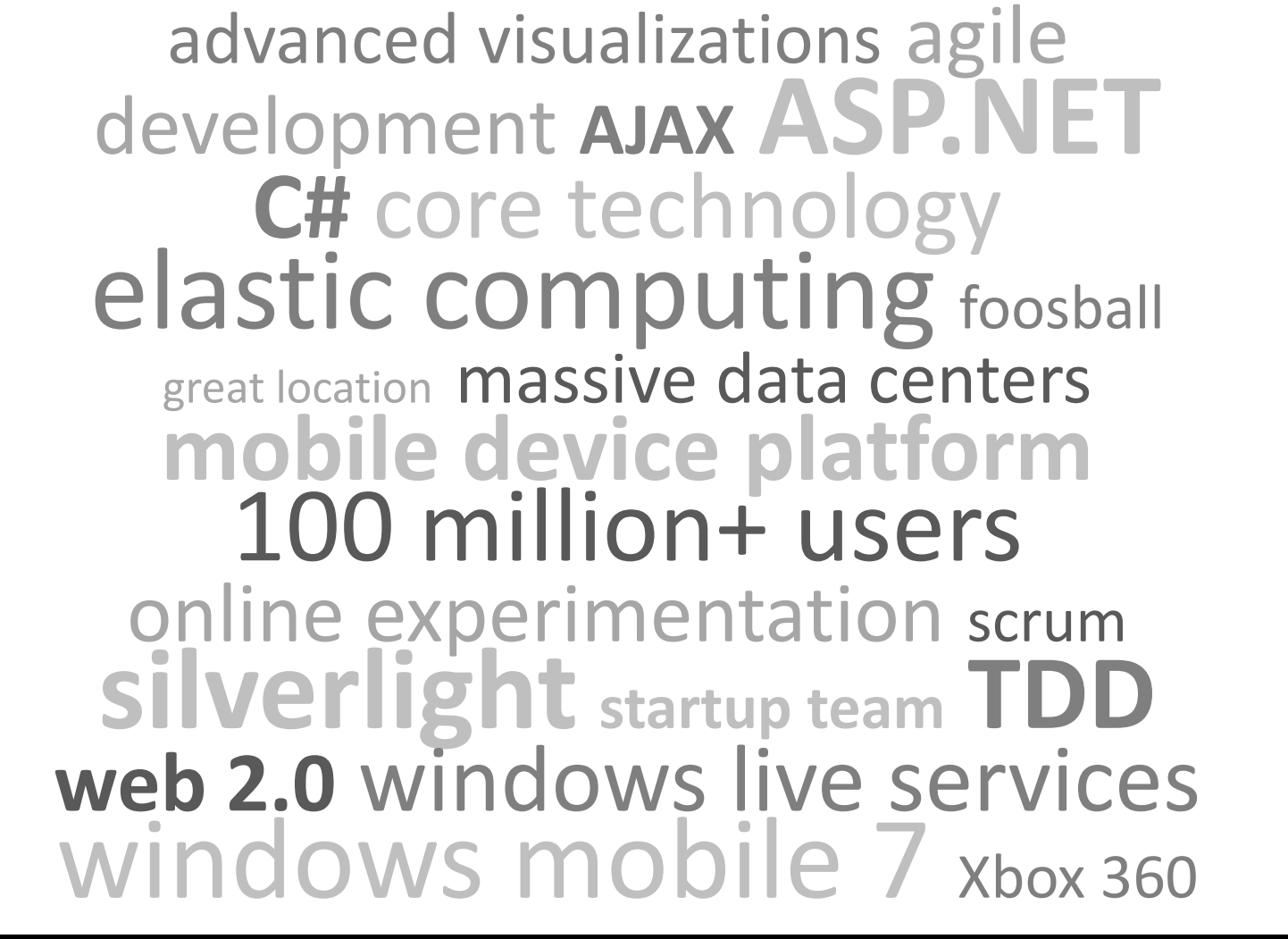
Test Management & Requirements
Experience & Tools

printed in Germany

free exemplar

www.testingexperience.com

ISSN 1866-5705



Add your tag to our cloud!

Microsoft's core tech development team in Europe is looking to fill key developer positions working on cutting-edge projects in Dublin, like online experimentation, datacenter "edge computing", and Windows Mobile 7.

Read job descriptions, watch videos, and learn more at **www.joinmicrosofteurope.com**.

Send your CV to **eurojobs@microsoft.com**.

join microsoft europe.com

Core tech development jobs based in Dublin

Editorial

There is no road, the road is made by walking. The words of my compatriot the poet of the Generation ’98 Antonio Machado is the leitmotiv for the “testing experience”. The first issue achieved around 30.000 readers worldwide. We want to thank you, the community, for giving “testing experience” the support.

After the first issue we got reams of congratulations from all around the world. We got also some valuable hints that we want to implement in this and next issues. Thanks to all of you for the support and your time. We can only be as good as we should, if we get your support and hints.

After sending the first emails our server had a lot of problem to respond all the requests in an adequate time, sometimes the server “went to knees”! Sorry for that. We knew that it will be some gut respond, but really we never thought that “testing experience“ would get that kind of respond. Afterwards I think that the market was missing something. We hope to fill the gap and fulfill your expectations.

I would like to thank again the authors of the first issue. We got a lot of positive messages regarding their articles and opinions. We got it! Gracias!

The second issue based on “Test Mangement and Requirements” has brought together old and new authors with opinions, trends and facts on these topics. We thank all of them. I’m looking forward for your opinions and valuable hints. Don’t doubt to send it to us. We appreciate it very much.

We want to thank all the sponsors of the “testing experience”. There are a lot of you linking it in to your website, blog or email signature.

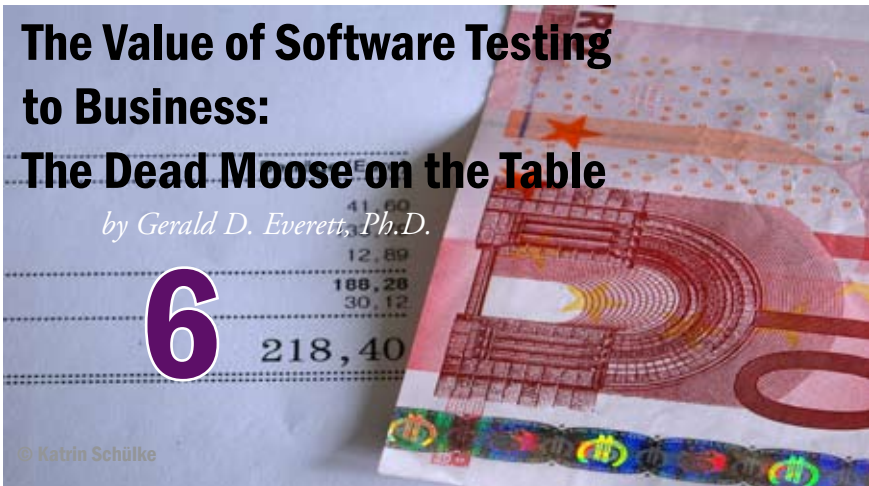
Last but not least we want to thank the companies and organizations, which advertise, for believing in the road, we make by walking.

Yours sincerely

José Manuel Díaz Delgado

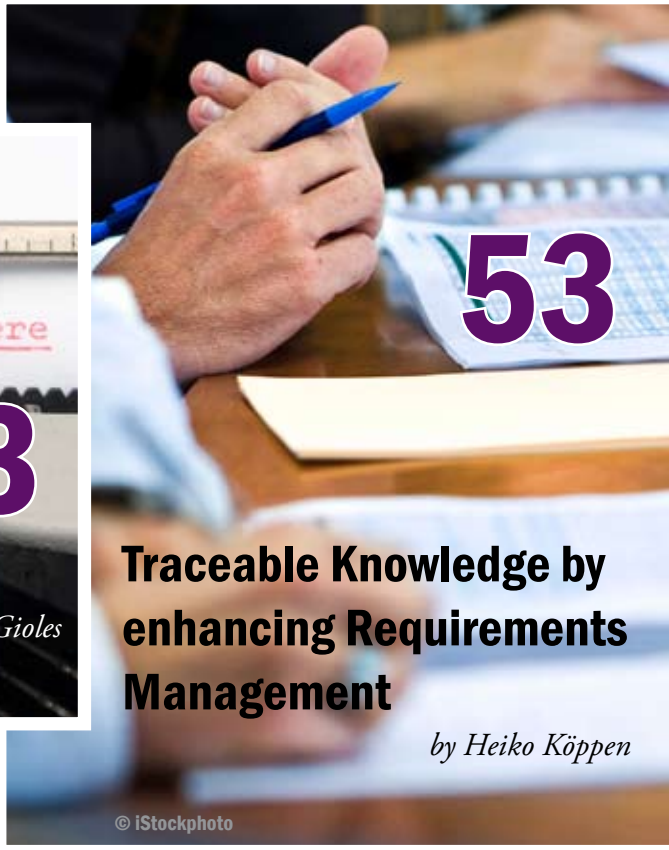
Content

testing experience



Requirements-Based Testing – Ambiguity Reviews

by Gary E. Mogyorodi, B.Math., M.B.A.



Traceable Knowledge by enhancing Requirements Management

by Heiko Köppen

Testing Web Services with TTCN-3

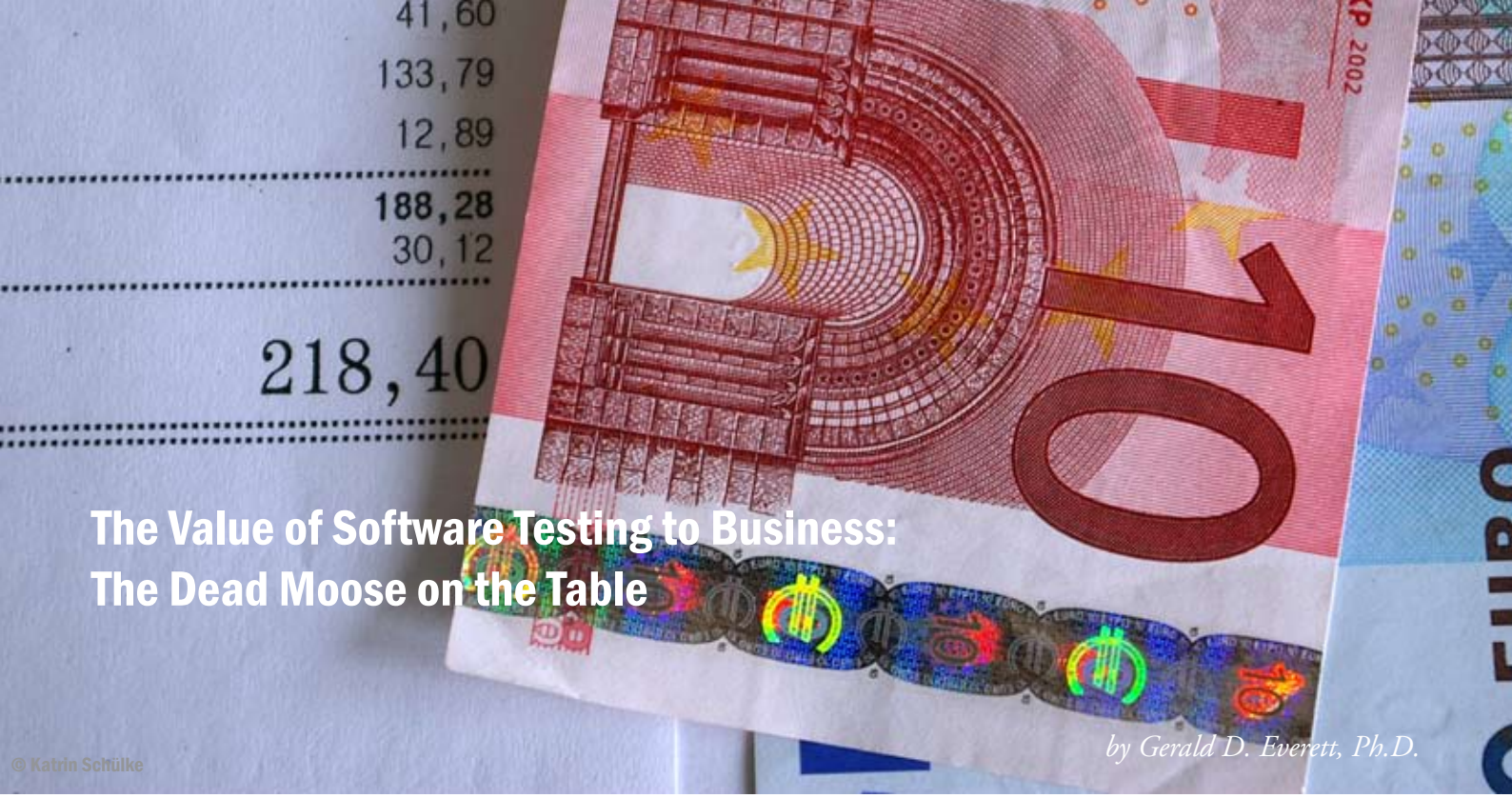
by Theofanis Vassiliou-Gioles

The Future of Software Security

by Prof. Dr. Sachar Paulus

Requirements and test management: More safety in vehicles through traceable testing

by Norbert Ruck, Ralf Lulei, Thorsten Geiselhart and Gerd Rockweiler



The Value of Software Testing to Business: The Dead Moose on the Table

by Gerald D. Everett, Ph.D.

Article Goal – to initiate discussion in the testing profession community that culminates in a concise, measurable definition of the value of software testing to business.

What A Dead Moose Looks Like

Have you ever had this experience as a software test manager or leader? The software development project is down to the last few weeks before the software is due. The testteam has identified some defects in the software that the testers have suggested are "go live" risks that could be mitigated by further testing, further corrections, further …. The business sponsor for the software says in effect, "yes, that is a risk, but it is one I am willing to take in order to achieve our (fill in the blank with "revenue", "sales", "customer", "product", "service", etc.) goals *this quarter by delivering the software when scheduled*. Later in the middle of the disaster caused by inaction on these risks, the typical witch hunt proceeds to place blame on everybody in the project … except the person who agreed to take the risk in the first place. This witch hunt in the presence of the guilty party while ignoring the guilty party is what the Canadians humorously call a "dead moose on the table".

The Real Dead Moose Dilemmas

I really see two dilemmas in the above scenario that the software testing profession needs to address. I believe that these two dilemmas are the testers' moose on the table. Both dilemmas revolve around what I consider a test professional's current inability to articulate the business value of software testing to business professionals.

The first dilemma is a question of how to gain sufficient credibility with a business sponsor so that the sponsor doesn't ignore real, identifiable risks as the development project approaches the due date. With a strong statement of testing business value in front of them, business sponsors would be much more hesitant to assume high testing-identified risks and much more disposed to finding more time, resources, budget, or whatever is needed to substantially lower those risks.

The second dilemma is the first dilemma driven up to the top of the food chain. When a new software development project is started, a budget and schedule for testing is included more often than not. That is a significant victory for testing over the 1990's when testing was done "only if we have the time". The second dilemma concerns how the testing budget and schedule are treated when the development team gets into scheduling trouble. When the developers run out the clock on the project, the project managers consistently use the testing schedule as the on time/on budget parachute. As with the risk-taking decision, as long as software testing professionals cannot articulate the value of testing to the business, then testing and its results will always be viewed as expendable.

I think it is time to solve these two dilemmas with an open discussion by testing professionals about the business value of software testing. I would like to start the discussion by some thoughts for your consideration. Hopefully you will reply to the article with your experience and insights.

Considerations for a Software Testing Value Proposition

There are several pieces to the software testing value puzzle in hand already. I'll offer what I know. I challenge others reading this article to contribute their knowledge and experience to complete the puzzle.

Software Testing Costs

There are a number of legitimate software testing costs during software development. These costs are frequently cited by business folks as a reason not to test. These software testing costs include:

- Test team salaries
- Tester training - basics
- A separate test computing environment
- Automated testing tools
- Tester training - tools

It is true that doing software testing (correctly or incorrectly) is expensive. I don't think this fact detracts from the probability of software testing having a greater value than cost to the business. It is just an incomplete story at the moment.

Software Failure Development Costs

There are a number of recurring software development costs associated with poor or nonexistent testing that I would expect to be part of a testing value statement and could offset the perceived testing costs listed above. All too often, a combination of organizational structure and internal accounting practices completely mask these costs from reconciliation with their effect on testing completeness. These software development costs include:

- The cost of code so badly written that redesign/refactoring is required
- The cost of correcting code before the software is released to end-users
- The cost of correcting code after the software has been released to end-users

Part of the reason these costs are masked from

view is a popular basis of developer incentives. All too often, bonuses, raises, or promotions are based mostly on "how many lines of code did you write?" In my 40+ years of computer professional experience, I have never encountered a company that took the "carrot" approach by incenting developers to write better code by saying, "your code for application X had the fewest defects of any other developer; therefore, you get a bonus for this year's work and a raise to continue good work next year." Similarly, I have never encountered a company that took the "stick" approach by incenting developers to improve their code by saying, "your code for application X had the most defects of any other developer; therefore, you do not get a bonus for the year or a raise for next year. On the other hand, we will help you improve toward a possible bonus next year by paying for additional programming skills training."

Why all the fuss about the cost of correcting code anyway? If you subscribe to the industry research by Basili and Boehm, the fuss is enormous. [1] For example, if your development programmers caused 1,000 defects that had to be corrected during the system testing phase, then according to Basili and Boehm your programming staff just cost your company more than $1M USD in correction effort. I have seldom seen a development project budget with $1M USD allocated in advance for corrections. The story gets worse. If the same development programmers caused 1,000 defects that had to be corrected *after* the software was released to customers/end-users, then according to Basili and Boehm your programming staff cost your company more than $14M USD in correction effort. The inescapable conclusion is that sloppy programming can cost a project more money to correct defects than the original budget to develop the software, whether those developers are in-house employees, contractors, or off-shore workers.

So the cost of correction must somehow become one of our software testing value puzzle pieces. This puzzle piece presents a particularly difficult challenge to business because the cost of correction during development is seldom reported; furthermore, the cost of correction after development is usually billed to a separate maintenance organization. Seldom does a company attempt to reconcile these support costs back to the responsible development project team.

Software Failure Business Costs

There are at least two business costs associated with software failure after its release. These costs are in addition to the costs for correction just discussed. These business costs include:

- Primary business failure costs - Internal business failure in daily operation
- Secondary business failure costs - Business failure that causes customer business failure

Primary business failure costs are those costs of lost business because the software stopped daily business activities. This cost is usually expressed as revenue loss or customers loss, very measurable quantities. A not too recent but relevant example is Toys R Us. This company decided to hop onto the internet as quickly as possible to capitalize on the vast market the internet promised instantly. The venture was a business disaster. Within 12 months, Toys R Us appeared on the web, lost the company $9M USD in expected on-line sales, and went off the air. The Toys R Us website contributed directly to a $14M USD loss ($9M USD in lost sales and $5M USD in band-aid effort to salvage the hastily constructed website). [2]

I was one of the sales that Toys R Us lost during that period. So I experienced first hand the shopper-unacceptable slowness of response time and deadly software failure that contributed to the website's demise. Performance issues notwithstanding, there was a defect in the purchase screen, detectable by ISTQB testing methods, that I suggest killed the Toys R Us internet business. Let me describe the defect and you can judge for yourself.

I was able to collect about $250 USD in Christmas toys for my grandchildren in my Toys R Us shopping cart. When I attempted to check out (purchase the toys), I encountered the following error on my billing screen, "city field required – please enter a city name." I had entered my city name, San Antonio. I was baffled. So I put on my tester hat and tried for a half hour to enter something in the billing screen city field that would be acceptable. I failed.

I cancelled my order because I couldn't get past the billing screen address field defect. Knowing the challenges in deploying a new website, I decided to be a good Samaritan and tell Toys R Us about their billing screen problem. I clicked on the Help button. In the space provided, I objectively documented the billing screen defect. Then I clicked on the submit button. Another screen popped up asking me my name and address. When I completed the screen and hit the send button, I received the same error message, "city field required – please enter a city name." By reusing the same, untested city field validation code, the Toys R Us programmer had managed to block ALL purchases and ALL customer comments that might have alerted Toys R Us to a source of their lost business. The business loss in this case was the entire on-line business, the whole enchilada. Just to tell the rest of the story, in 2002 Toys R Us reappeared on the internet via Amazon.com which is a consummate internet retailer.

Secondary business failure costs are those costs of lost business incurred because the software stopped daily business activities of that business' customers. This cost is much more difficult to assess than those of the primary business failure because the loss varies with the customers. Here are two examples of secondary business failures.

Recently, Blackberry experienced a massive, preventable computer system failure that took it off the air for two workdays. [3] Blackberry is an internet messaging service that currently has 8M customers worldwide. The outage affected only the North American customers (Canada, the USA, Mexico), maybe 4M customers. The burning question is, "how much business did 4M customers lose while they were unable to transact business with their Blackberrys?" The answers could range from "not much impact, I was on vacation and just checking in with the office" to "I lost a $10M USD contract bid because I was unable to respond by the deadline." Somewhere in between lies the truth … and a piece of the software testing value puzzle.

Another example is last year's preventable crash of TurboTax that absolutely stopped the submission of over one million Internal Revenue Service (IRS) Tax Returns in the USA just before the tax filing deadline. [4] There is an unlikely white knight in this story, the IRS. For the first time since it was founded in 1935, the IRS extended the tax filing deadline to allow a private business, the TurboTax system, to recover and submit all the returns halted by the crash. The age of miracles has not passed!

From the testing value perspective, I think it would be interesting to calculate the potential IRS penalties TurboTax customers would have been fined if the IRS had not intervened with a grace period. This is one time the secondary impact could be calculated to the penny. The IRS has a standard schedule of penalties for submission after April 15 based on the amount owed on the late tax filing and the lateness of the filing past the deadline. TurboTax had captured every late tax filing before their crash.

Using the IRS penalty schedule and the TurboTax copy of the tax filings, an exact penalty could be calculated for each late submission and the penalties totaled for the true cost of the crash to TurboTax's customers. It is easy to speculate that TurboTax's greatest fear was a class action lawsuit by their customers to at least recover all penalties that they owed the IRS due to TurboTax's crash, at most a class action lawsuit over TurboTax's guarantee to submit tax returns in a timely way. Since jail time for delinquent income tax submissions is possible under the IRS law, this particular secondary business loss just keeps on giving.

A Draft Value Statement

We have identified a number of development and business costs that can offset testing costs in software development. I propose that we begin to build a business case for software testing value from these costs. Consider the following equation as a starting point.

```
TV =    50% S DCDP * SCCED
      + 50% S (DCDP * DCCP)
      + SBLR
      - TBL
```

where	TV	is the Testing Value
	DCDP	is the number of Defect Corrections per Development project Phase
	SCCED	is the project Support Correction Cost per End-user Defect
	DCCP	is the average Development Correction Cost by project Phase
	SBLR	is the Software Business Loss due to testing Risks this project's business sponsor assumed in lieu of correction and, in fact, the risk occurred.
	TBL	is the Testing Business Loss due to incomplete test planning or test execution that allowed business loss to occur.

What I want us to be able to say to the business sponsor of my software development project is, "the testing budget for this project is $500K USD; however, the expected value of this project's testing to the company is $23.6M USD.

Here is my rationale for the testing value equation. The additive expression "50% S DCDP * SCCED" represents the value that testing brings to the development team by reducing the number of defects ultimately delivered to the end-users. Clearly, it costs more to correct software in the end-user's hands than on the developer's workstation. Basili and Boehm underscore just how disproportionate those costs are. The only point of contention is the percentage of those development defects that would have reached the end-user without testing. Based on my experience, I suggest 50% to be a conservative but appropriate multiplier of the defect costs because many, but not all defects discovered during development will directly affect end-users if not corrected.

The additive expression "50% S (DCDP * DCCP)" represents the value that testing brings to the support team by reducing the number of end-user discovered defects that the support team must correct. Using the cost of correction by phase in which the defect is discovered, the value of saving support effort is graduated by how early in the development process the defect was discovered. I suggest a 50% multiplier because I know it isn't 0% and it certainly isn't 100%. I'll start the discussion by splitting the difference. Suggestions based on your experience are welcome.

The additive factor "SBLR" represents the consequences of a business sponsor "taking a risk" that has been identified by testing and recommending the wrong action against that risk. This factor probably represents a challenge for the organization, because it requires someone to make a connection between the software development project risk decisions and the HelpDesk calls and the customers who suffered the business loss because of a poor risk gamble by the business sponsor. Although this factor could be an open-ended loss, it will have more credibility with the business community if we limit the losses just to those within the first year after release.

The expressions and factors described so far accrue to the benefit of testing. They represent value that testing uniquely contributes to software development. In business, there is always some kind of offsetting entry on the books, because there is no such thing as a free lunch. In the case of testing value, consider the factor TBL to be that offset. "TBL" represents business loss due to poor or incomplete test planning or execution or both. As with the business loss due to business sponsor risk taking, this cost will be challenging to determine. It will require a comparison of end-user detected defects with the project's test plans and test execution results to identify candidate defects missed by testing. Then the business loss due to those defects must be determined.

Here is one last thought about the test value equation. The cost of the software development project is not included anywhere in the equation, because I believe that testing value needs to be predicated on what testing accomplishes for the project regardless of the project's budget.

A Test Drive of the Test Value Equations

Here is how I derived the example testing value I quoted in the previous section. I challenge you to collect and share real metrics from a recent software development project and see how large a value your data places on testing. Your mileage may vary.

Several of the factors are inter-related and can be derived together. This derivation relies on several management practices that may or may not be implemented in your organization. Those practices are:

- Test Management that includes full test planning documents and test execution results reports (ISTQB recommended).
- Defect Management that includes a complete defect log for the project (ISTQB recommended).
- Project Management that includes the actual cost of defect correction by development phase.
- Project Management that includes project documentation of all risks identified during the project and their resolution at project end.
- Support Management that includes complete end-user/customer problem log for the software system delivered by the development project.
- Marketing Management that includes a survey of business impact on the company's customers when any of the company's software systems crash.

I will start with the defect log maintained by Test Management. The defect log will tell me how many defects were discovered and corrected during each phase of the development which is S DCDP. The Support Management average cost per defect correction is SCCED. I will then apply the Support Management average cost per end-user defect correction to the defect log numbers to evaluate S (DCDP * SCCED).

The Project Management average cost per defect correction per phase is DCCP. I will then apply the Project Management costs to the development defect log numbers by phase to evaluate S (DCDP * DCCP).
If the development project did not keep a defect log, then you are relegated to a SWAG (scientific wild ass guess). One reasonable SWAG is using a Rayleigh curve to estimate the number of defects that should have been found in the project's software. Group the total defects into the four general development phases identified by Basili and Boehm. Multiply each Rayleigh group total by Basili and Boehm's industry average cost to correct defects during that phase.

Here is an example Rayleigh curve for an 18-month project that developed a software application containing 3M lines of code with an average of 1 defect per 1K lines.

Testing Days Germany 2008

Compuware and Diaz & Hilterscheid invite you to attend a free event series for testing professionals in autumn 2008:

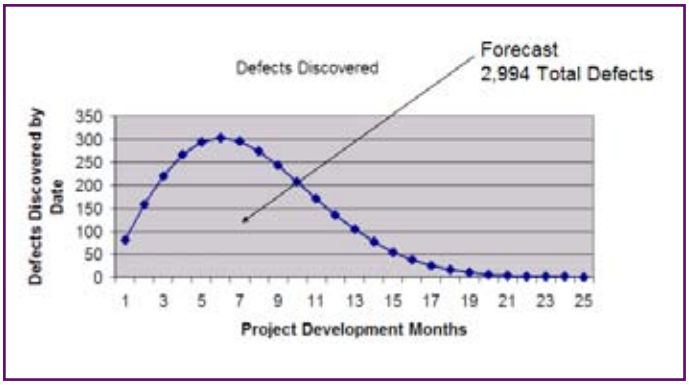
- **26th September Duesseldorf**
- **29th September Frankfurt**
- **30th September Munich**
- **1st October Hamburg**

Presentations include a mini tutorial on test design by „Mr. Test“ Hans Schäfer, case studies and many more. Events will be followed by a get together for testing professionals to exchange experiences.

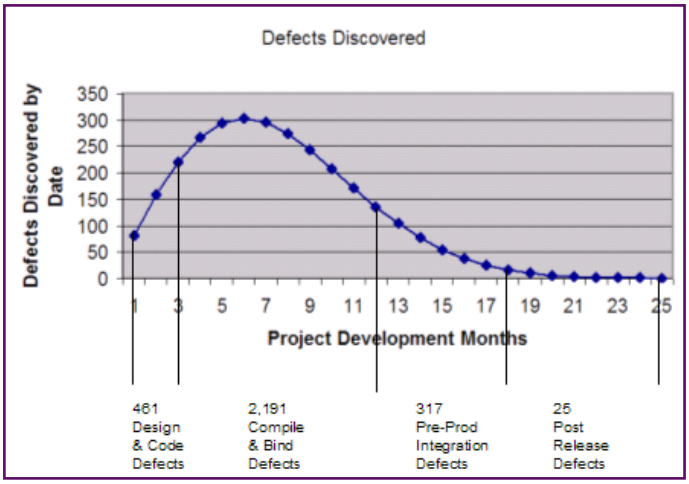
More details to follow soon on
www.compuware.de/events

Save the date!

COMPUWARE®



Here is the Rayleigh curve showing approximate grouping of defects by Basili and Boehm's development phases.



Here are the Basili and Boehm's cost factors per development phase.

- Design & Code 461 defects @ $139/correction
- Compile & Bind 2,191 defects @ $455/correction
- Pre-Prod Integration 317 defects @ $977/correction
- Post Release 25 defects @ $14,102/correction

If you have not used a Rayleigh curve to forecast defect discovery, please see Chapter 12 of Software Testing: Testing Across the Entire Software Development Life Cycle for a suggested non-statistical technique. [5]

Now we are in a position to calculate the test value expressions.

50% S DCDP * SCCED =
50% (461+2,191+317) * $14,102 = $20.9M

50% S (DCDP * DCCP) =
50% (461*$139 + 2,191*$455 + 317*$977) = $0.7M

To get the business losses due to ignored testing risks (SBLR), you will need to review the project teams closeout report to identify the disposition of testing risks raised duing development. List out the risks "taken" by the business sponsor. Next, review the Support team's list of customer problems by application or system. Match up the customer problems for the development project you are analyzing. Narrow the list of customer problems to those apparently caused by defects discovered by testing during development but not corrected because the business sponsor "took the risk."

You will need to contact Marketing to see how best to obtain the total cost of the customer problems for which the risk occurred within the first-year after development was done. Some business losses due to software problems could equate to $5,000/day while other business losses could equate to $5M/hour. Arbitrarily, I will use a number that I consider a moderate first year business loss for large software development projects that are mission critical or near-mission critical to the end-users.

SBLP = $4M

To get the business losses due to incomplete test planning or execution (TBL), you will need to follow a similar path through the project results. Start with a review of the Support team's list of customer problems by application or system. Match up the customer problems for the development project you are analyzing. Subtract from the list of problems those that you assigned to the business risk loss because once the test team has identified a risk to management, the test team's cost responsibility has been fulfilled. Categorize the remaining list of customer problems by possible cause. Match those problem categories to the test plan from Test Management. Identify those problems that fall well within the test plan but were not discovered by subsequent test execution.

You will need to contact Marketing to see how best to obtain the total cost of these customer problems caused by missed testing within the first year after development was done. Arbitrarily, I will use a number that I consider a moderate first year business loss for large software development projects that are mission critical or near-mission critical to the end-users, and the test team has completed some form of professional skills certification like the ISTQB.

TBL = $2M

There is an interesting footnote to the process for deriving the TBL factor that I hope you already caught. When you reduce the total list of problems for this application by those problems that fall well within the test plan but were not discovered by testing, the remaining problems are those that occurred outside the test plan. Perhaps a review of this list of "outliers" could indicate ways you can improve your method and effectiveness of test coverage.

We now have all of the equation expressions evaluated and have determined all the remaining factors. Here is the resulting business value of testing for our example business.

TV = $20.9M + $0.7M + $4M - $2M = $23.6M

Conclusion

We come back to the dead moose on the table. By developing an equation for measuring the value of testing to business, we can carve up the carcass into pieces small enough to assign ownership, declare business benefit or liability, and be permanently dragged off the table.

Please send your observations and suggestions for the article's proposed test value equation to jerry.everett490@gmail.com.

References

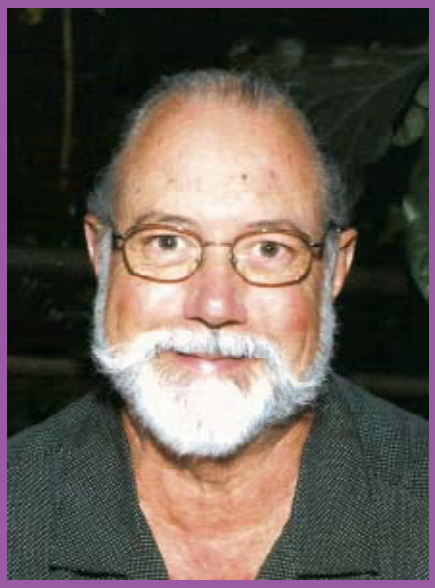
[1] Basili and Boehm, "Software Defect Reduction Top 10 List", IEEE Computer Society, vol. 34, (No.1), January 2001, pp/ 153-137.

[2] "Spending on Web Store Hurts Toys R Us Profit", Stephanie Stoughton, Washington Post Archives, Aug 17, 1999, p E1.

[3] http://www.origsoft.com/Reading_Room/nightmares.htm

[4] http://www.irs.gov/newsroom/article/0,,id=169583,00.html

[5] Everett and McLeod, *Software Testing: Testing Across the Entire Software Development Life Cycle*, Wiley-Interscience, 2007, Chapter 12, ISBN 978-0-471-79371-7.

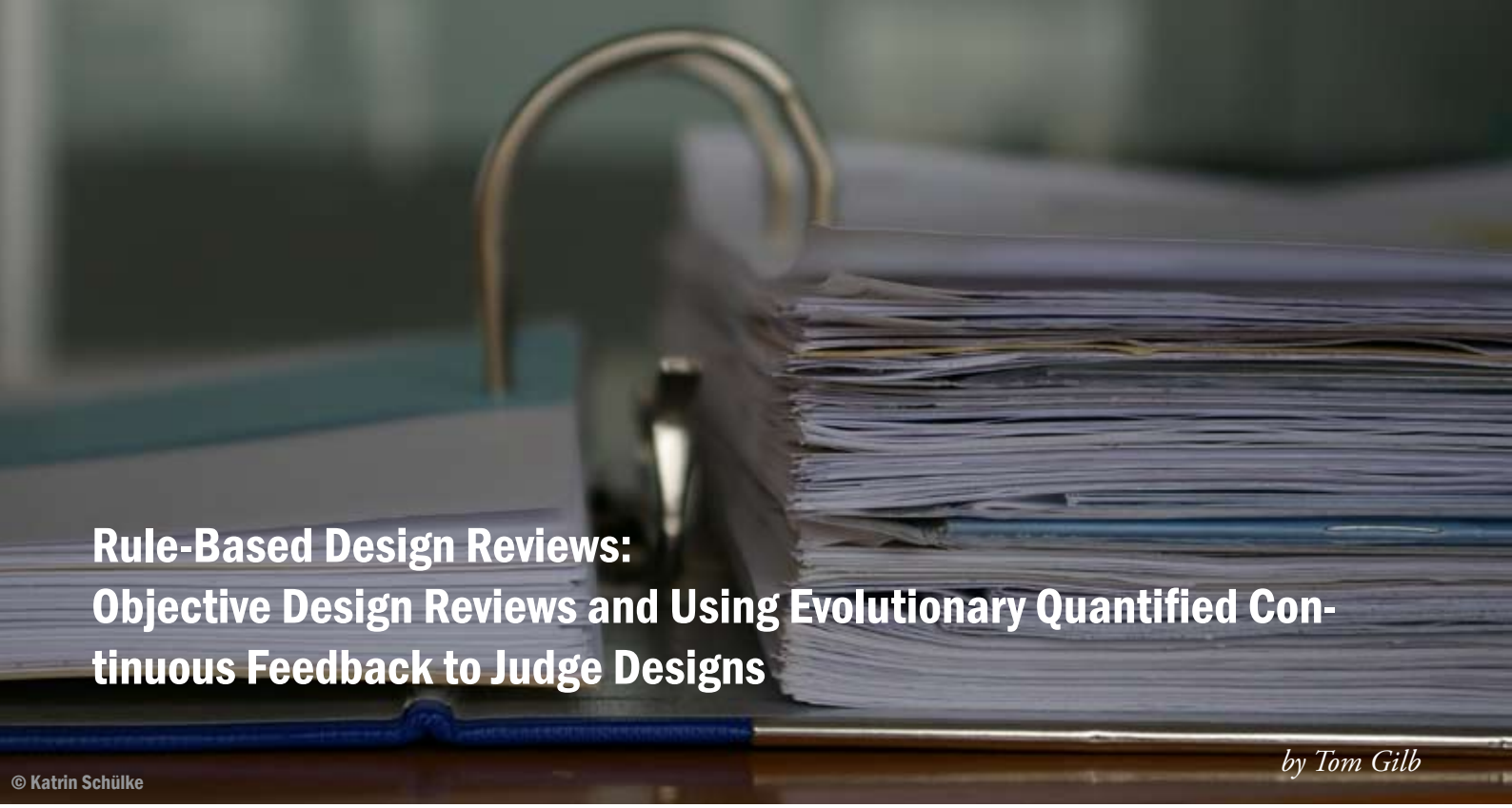


Biography

Gerald D. Everett, Ph.D. has been designing, developing, and testing software systems for more than 40 years. Paralleling his corporate software development work, Jerry has spent 10 years as a university faculty member in Computer Sciences and Management Information Sciences departments.
For the last 8 years, his career has focused on developing and delivering courses in software testing to IBM testing practitioners and consultants. Jerry's efforts have resulted in 21 formal software testing courses ranging from 8-hour introductions to 40-hour workshops. His love of travel has been fulfilled by IBM assignments to deliver these testing courses in Amsterdam, Austin, Atlanta, Bangalore, Beijing, Boston, Brussels, Budapest, Hersley, Hong Kong, Lisbon, Montpellier, Rome, San Francisco, Stuttgart, Tokyo, and Toronto. Paralleling his corporate software testing education delivery, he continues to be invited to deliver an average of 6 testing lectures per year to US universities.

Dr. Everett published Software Testing; Testing Across the Entire Software Development Life Cycle in 2007 which is enjoying adoption primarily by graduate level university courses. He is on the Board of Directors of the ASTQB and recently achieved his ISTQB Foundation Level testing certification by using only the ISTQB syllabus and his textbook for study materials.

Jerry's newest professional interest is researching and piloting effective ways to teach software testing in virtual social worlds like Second Life.



Rule-Based Design Reviews: Objective Design Reviews and Using Evolutionary Quantified Continuous Feedback to Judge Designs

© Katrin Schülke

by Tom Gilb

Abstract

Design reviews would benefit from the support of formal rules. By the use of relevant rules, it should be possible to ensure *prior* to a review that all the relevant information for a review is present in the design specifications, and that all the minimum review criteria are met. This will ensure management time is not wasted and aid better decision-making. This paper recommends that the Specification Quality Control (SQC) method be used to do this additional quality control. In addition, this paper outlines the impact of Evolutionary Project Management (Evo) on the design review process.

Introduction

We know that most projects are a partial or total failure, and few are totally successful (Morris 1994, Taylor 2001, Johnson 2001: Extracts from Extreme Chaos 2001, a Standish Report). I suggest that a key reason for this lack of project success is the failure of design reviews. In other words, whatever designs did get approved, should probably not have been.

Based on reading the literature, and participating in a very large number of requirement and design inspections in many different industries internationally, I fear that most design reviews are carried out:

- on poorly written design specifications (in the sense that the design specifications lack sufficient detail for them to be safely evaluated), and
- using highly polluted requirement specifications (Requirements being the basis for evaluating any design).

I find it unacceptable that a design review is given *no quantified knowledge* of the quality of the design specification, or of the estimated ability of the design(s) to impact on the requirements (that is, the system performance and costs). The underlying problem is that specification of design is carried out on the basis of inadequate design standards.

It is the point of this paper to suggest the following remedies:

- a highly defined standard of requirements is met before entry into the design process itself is permitted
- design specification should initially pass quality control (SQC) against design specification rules to make sure the designs are clearly and fully described
- designs should be specified in sufficient detail to enable reasonably accurate estimation of their dominant performance and cost characteristics
- a set of designs should be seen to credibly and numerically contribute to meeting the requirements (the set of performance targets within the resource budgets)
- the design review process should work in the context of evolutionary cycles of design (for example, in 50 steps), and not operate on a large monolithic total design set

Terminology

Specification Quality Control (SQC): SQC is also known by the name 'Inspection.' Given that 'Inspection' has a different meaning to engineers within manufacturing, I prefer to use the term SQC. SQC is also sometimes termed 'Peer Review' I do not use this term because I want to make a clear distinction between 'quality control' and 'review'.

SQC is a rigorous quality control discipline concerned with defect detection, defect measurement, defect removal, process improvement and entry/exit controls. It is based on evaluating specification conformance to specification rules.

Traditionally, SQC does not pretend to judge the specifications in terms of their relevance or profitability in the real world. It is primarily concerned with ensuring that the specifications are clear, complete and consistent by checking a specification and any of its source and kin documents against 'specification rules'. It judges whether the specification is suitable use in subsequent engineering or management processes. However, by using a different type of rules, 'specification review rules', it is possible to extend the SQC process to checking the readiness of specifications for review. This could be applied for a business review or a technical review.

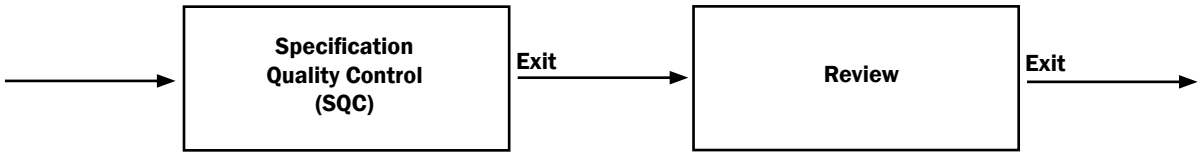


Figure 1: The two necessary distinct processes:
- Specification Quality Control (SQC) – Is it following the standards (rules)?
- Review – Is it the right stuff?

Review: A review is any process of human examination of ideas with a defined purpose and defined standards of inquiry.

Design Specification: A design specification is the written specification of a design idea. A set of design specifications attempts to solve a design problem. Identification and documentation of the individual design ideas and their potential contribution towards meeting the requirements help in the selection of the 'best' design ideas for implementation.
The design specifications should contain information about the expected attributes of the designs for meeting requirements. This 'expected attributes' information of a design specification might be in the form of an impact estimation table or, it can be as simple as an assertion of impacts on requirements, referenced by their tags (see example in Figure 2).

Engineer Motivation:

Gist: Motivate and, use of free time off.

Type: Design Idea.

Impacts [Objectives]: {Engineering Productivity, Engineering Costs}.

Impacts [Costs]: {Staff Costs, Available Engineering Hours}.

Definition: Offer all Engineers up to 20% of their Normal Working Hours per year as discretionary time off to invest in Health, Family and Knowledge {Studies, Write Papers, Go to Conferences}.

Source: Productivity Committee Report 1.4.3.

Implementor: Human Resources Director.

Figure 2: Example of a design specification showing some impacts (though no specific numeric estimates!) for the design on the requirements.

I will present this paper in terms of a set of design review principles.

DR1: If the specifications are unclear, you are not ready to review whether the design is the 'right thing' to do.
DR2: You cannot review design if you are unclear whether a design is mandatory (a design constraint, which is a 'requirement') or optional (one of many possible design solutions).
DR3: Design impacts must be calculated and submitted to a review, they cannot systematically be developed *during* the review.
DR4: It is part of the purpose of a design review to identify any defective information in the design impact *estimates*.
DR5: Certain objective review criteria must be met prior to actually carrying out a review; otherwise the review may be wasted.
DR6: The review process should not wait until a large amount of specification is completed; sampling reviews should be held early and frequently to spot systemic problems early.
DR7: A design review process should be carried out on a realistic amount of information and be conducted at an effective rate. Don't overwhelm the reviewer, so they become incapable of spotting problems.
DR8: In order to benefit from feedback, the design review process should be done evolutionarily, as a long series of design reviews: each review deciding on the next planned evolutionary step.
DR9: The real purpose of design reviews is not to approve a design as correct, but to uncover specific risks and ignorance.

Principles

Principle DR1: If the specifications are unclear, you are not ready to review whether the design is the 'right thing' to do.
Have you ever actually counted the quantity of unclear and ambiguous expressions per page in a real specification? I get my clients to do this using specification quality control (SQC) almost every week. The result is consistent and always provides an element of shock.
I define a specification defect as any violation of a set of specification rules. As a simple introduction to SQC, I ask a team of two to five people to select a random page of their 'typical' specification and then to spend about 15 minutes checking it individually, applying these two rules:

Rule 1: The specification must be unambiguous to all in the intended readership.
Rule 2: The specification must be clear enough to test for correct implementation.

For the first rule, it does not matter whether the people looking for defects themselves understand the specification; they must role-play the weakest link in the set of people who might have to correctly understand it (for example, 'Imagine you had just started here right out of school', or 'Imagine you were an Indian subcontractor in Bangalore').
I also ask the participants to judge whether the rule violations they find (the 'specification defects') are 'major' (or minor). Defects are 'major' if it is judged *possible* (as opposed to *im*possible) that the faulty specification *could* cause something wrong to happen during system engineering or in the operational system (for example, product faults, test case faults, procurement errors, estimation errors, software bugs, and delays).
Most people (for example the writers of the specification and their peers) consistently manage to find between 5 and 15 defects in a single sample page of a specification within the 15 minutes.
From experience, I know that for a small team of two to five checkers, if you double the number of majors found by the checker finding the most majors, then you tend to have the total number found by the entire team. I also know that the initial effectiveness in such a situation for finding majors is in the range of 25% to 35% (This is partly due to the time allowed, and partly due to the source information available). For the sake of simplicity, say 33% (a third).
So if 15 majors were found, I'd estimate that there were 15 x 2 x 3 = 90 defects on the single sample page. That means there are about 90 potential misunderstandings of a specification per page. This is 'normal' technical specification 'pollution', but it is not a good or necessary basis on which to decide if the design itself is a good one. Any one of those major defects alone is capable of corrupting our understanding of a whole design, and capable

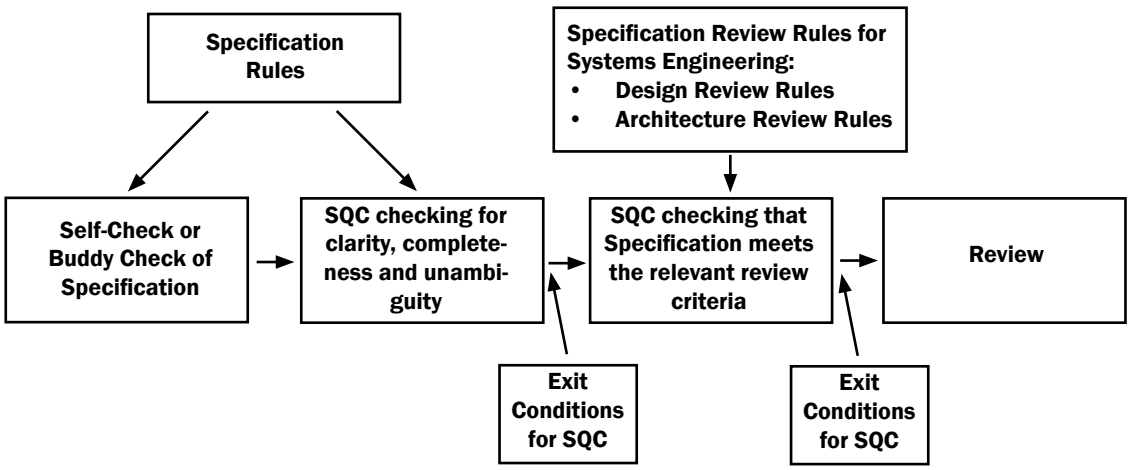


Figure 3: SQC and Review processes. Review should follow successful exit from SQC.

of making our judgments, in a design review, worthless.

The bad news is that this level of majors/page will continue to persist unless management 'changes the game'. The good news is that, if you want to, you can bring down the average level of majors/page by two orders of magnitude within weeks of doing the right things.
You can bring down the majors/page density by stating the specification rules (such as 'clear' and 'unambiguous'), and then training individual engineers to follow these simple rules. You do that by teaching the rules, and then making sure that rule violations are measured. Work is considered unacceptable when a defined level of pollution (such as 1 major/page) is exceeded. Under such conditions, an individual will learn at a rate of about 50% defect injection reduction per cycle of specify/check/correct. We proved this initially in 1988 at Douglas Aircraft for hundreds of engineers working on airplane drawings. We repeated it the year after at Boeing, and later we experienced this same improvement rate at Ericsson. It doesn't matter where or with which technology you do this, it works. Others, such as Raytheon Defense Electronics, have reported the same experience working within military software projects (Haley et al. 1995).

What does all this mean for design reviews? In summary, we need to quality control all specifications and make sure they are not polluted, before we can seriously undertake a design review to look at whether we are doing the 'right thing.'

I suggest that you need at least 2 entry conditions into a design review:

- all requirement specifications have successfully exited on a numeric basis from SQC
- all design specifications have successfully exited on a numeric basis from SQC

I recommend that you consider setting the numeric level at 'less than 1.0 estimated remaining major defect/page'. Anything less is wasting your review team's time and your project time. What do you do today instead?

Principle DR2: You cannot review design if you are unclear whether a design is mandatory (a design constraint, which is a 'requirement') or optional (one of many possible design solutions).
What most people title 'Requirements' often includes some design specification, which is not actually part of the requirements. Only design constraints should be specified as part of the requirement specification: any design that is optional has no place there, it ought to be only in the design specification. (In case you are wondering about this: requirements are the end states needed by stakeholders, irrespective of implementation detail. Design is a decision about how to implement those requirements in practice.)
Of course, mandatory designs (the required design constraints) must also have detailed design specifications. You need to ensure you have the design specification for any mandatory design (clearly marked as mandatory), and that it is considered alongside the optional design.

Principle DR3: Design impacts must be calculated and submitted to a review, they cannot systematically be developed *during* the review.
There is no time in a design review to begin a process of collecting facts and making estimates about the multiple impacts on performance and costs of each design. The proper time to do that is before the review, not during it.
If the estimates are not made, this fact should be caught in the SQC preceding the review. Note that such SQC merely observes that such estimates are not made, or not made properly

(for example, they lack any evidence).
So, tell me, do your current design documents have integrated, or referenced, the impact estimations for the designs, ready to feed into the design review process?

Examples of Specification Rules for Design

AR1: Cost Detail: The architecture must be specified in enough detail to at least permit correct order of magnitude impact estimation for costing.
AR2: Cost Estimates: Estimates must be made and included as to the order of magnitude of all critical costs of the architecture (particularly for those resources that are budgeted requirements).
AR3: Performance Detail: The architecture specification must include enough detail to allow order of magnitude correct estimation of the specification's impact on ALL specified performance goal levels (that is, all the work capacity, qualities, and savings requirements).
AR4: Performance Estimates: Estimates will be included for the impacts on ALL the critical performance requirements, at correct order of magnitude.
AR5: Background Detail for Estimate: Each impact estimate must be supported by:
the factual experiential evidence for the estimate,
the source of these facts,
the uncertainty boundaries/error margins (±%), and
a credibility rating (on a 0 to 1.0 scale).
These data ideally will be presented on an impact estimation table.
AR6: Additional Data: The architectural specification must include additional specification as detailed in the current architecture specification template.
This will include Stakeholders, Detailed Design Reference (if any), QC level, Review Approval level, Risks, Issues, and Dependencies.

Figure 5: Some Examples of Design Specification Rules. These would be used in a SQC of a design specification. Rule AR6 would likely be expanded into several distinct rules. Only if the defect level was sufficiently low (say, less than one remaining major defect/page would the specification be submitted for further SQC to see if the specification was ready for review (See Figure 6).

Principle DR4: It is part of the purpose of a design review to identify any defective information in the design impact *estimates*.
The design reviewers are allowed to question the accuracy of the impact estimates, and the evidence and credibility claimed. The design reviewers are experts in their fields and should consider if they agree with the data they are being presented with.
SQC carried out prior to a design review is mainly concerned with finding out that the required impact estimation process for designs was apparently performed.

Principle DR5: Certain objective review criteria must be met prior to actually carrying out a review; otherwise the review may be wasted.
In addition to making sure the design engineer has in fact adhered to the design specification rules (See Figure 5), there needs to be a check against design review rules in preparation for a design review. Only if the design meets the design review rules should a review proceed.

Examples (Incomplete) of Specification Review Rules for Design

AC1: Are the set of resource costs acceptable in relation to any relevant investment or operational budgets which exist?
AC2: Are the 'estimated architecture ideas performance impacts' sufficient to justify the resource costs? Are they the best impacts we can get at that cost level?
AC3: Does the suggested architecture, in terms of cost and performance impacts above, fit in with all other now envisaged architecture past, present, future. For example does the specified architecture cause us to exceed any resource budgets (time, money, space effort)? Does it threaten any critical performance level with unknown or known negative side effects?
AC4: Is this architecture specification the arguably best overall option for us? Have other promising options been evaluated? Should they be? Is there any dissent amongst expert architects on this matter?
AC5: Is there a plan for validating the real performance and cost impacts of this architecture in practice, before we commit to it on a larger scale?

Figure 6: Design Specification Review Rules to be used within SQC to check if the design specification is ready for design review.

Principle DR6: The review process should not wait until a large amount of specification is completed; sampling reviews should be held early and frequently to spot systemic problems early.
Once on a German Air Traffic Control project done in Sweden, I saw the signatures of 7 managers approving the detailed logic of the air traffic management: 40,000 pages of logic design (the next stage was the programming). Later that day, using the sampling process described earlier, they found 19 major defects in a random representative sample of 3 of the 40,000 pages of logic design. This was of course about one third of what was actually present in the pages (but for managers they were pretty good!). Their divisional director took 30 minutes to personally check the 19 majors – while the 8 of us waited in his office one evening - and agreed that they were serious. At about 20 majors per page present, that meant there were about (20 x 40,000) 800,000 majors approved for coding by the management review committee. I asked signee number 3 on the list why he signed off what we have now recognized was a very polluted document. His reply will not surprise the experienced reader: 'because the other two signed it before me." Now you know why I am skeptical about review committee approvals!
We had many an interesting discussion on the basis of this finding. The central one was that if they had bothered to do some simple samples early, like after 100 pages had been written, they might have been able to prevent most of this work from being totally wasted. In other words, if reviews had been carried out earlier, and if they had demanded numeric quality controls were in place, then it is unlikely that the defect injection would have been allowed to continue at such a level.
So there is a lesson about early partial work done sampling here. Don't wait for the entire specification before you do some basic quality control on it – a point we shall return to in this paper towards the end.

Principle DR7: A design review process should be carried out on a realistic amount of information and be conducted at an effective rate. Don't overwhelm the reviewers, so they become incapable of spotting problems.
If an attempt is made to review too great a quantity of design at once, then the analysis is unlikely to be done sufficiently, and the truth is likely to be obscured. If 40 or more (try 40,000 pages of, as in the example above) design specifications are delivered at once for design review, then the review group will clearly not have time to study, discuss, criticize anything in detail. Reality of risks and problems will be lost. It will not be possible for the review team to learn about their misjudgements. The project or product will probably fail and we will not be clear about the cause of failure.

I suggest that we need to feed only a small volume of ideas into a design review committee:
- in order to give the committee a chance to do its work properly
- in order to make sure that any preparatory work has been done properly

May I suggest one page of design per committee hour as a rough guide?

Principle DR8: In order to benefit from feedback, the design review process should be done evolutionarily, as a long series of design reviews: each review deciding on the next planned evolutionary step.
Ideally, most projects should be carried out using Evolutionary Project Management (Evo). The critical distinction between Evolutionary Project Management (Evo) methods and their generic cousins iterative (*we do cycles*) and incremental (*we cumulate*) is that evolutionary processes (which are also both iterative and incremental) gather facts about impacts, analyse those facts and change behavior, if necessary, in order to succeed in the higher level objectives (*we learn!*).
Evo means testing the stakeholder effects of a design idea in the field. We are suggesting maybe 50 ideas, one a week for a year of development effort. The design review commit-

tee becomes a learning process: that is, the review team will benefit from the Evo cycle experience feedback for implementation of each previous design; and they will learn quickly and realistically, from real experience, how to evaluate designs.

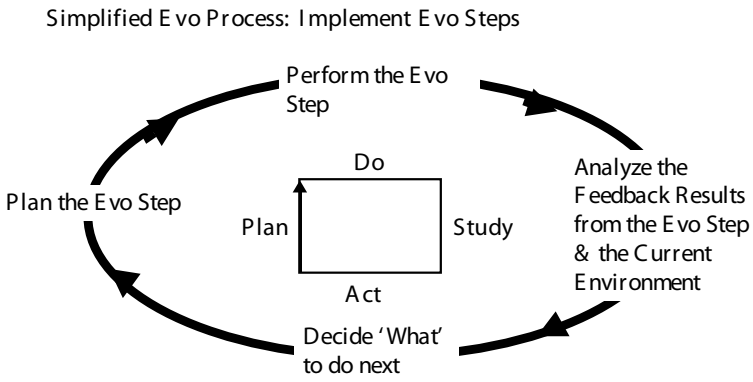


Figure 9: the Deming/Shewart PDSA cycle. The essence of Evolutionary project management and of consequent Evolutionary design processes.

Step Target Requirement	**Step 1** Plan % (of Target)	Actual %	Deviation %	**Step 2** to **Step 20** Plan %	Plan % cumulated to here	**Step 21** [CA, NV, WA] Plan %	Plan % cumulated to here	**Step 22** [all others] Plan %	Plan % cumulated to here
Performance 1	5	3	-2	40	43	40	83	-20	63
Performance 2	10	12	+2	50	62	30	92	60	152
Performance 3	20	13	-7	20	33	20	53	30	83
Cost A	1	3	+2	25	28	10	38	20	58
Cost B	4	6	+2	38	44	0	44	5	49

Table 1: This is a conceptual example. Three goals (performance targets) and two resource targets are having real (% of impact needed to reach the target) impacts on the performance and cost attributes, and are tracked as steps are delivered. The same IE table is also being used to specify the impact estimates for future planned steps. So at each step, the project can learn from the reality of the design impacts included in a step, and the deviation from design impact estimates. Designs and estimates can then be adjusted and improved from an early stage of the project.

Principle DR9: The real purpose of design reviews is not to approve a design as correct, but to uncover specific risks and ignorance.

Design reviews are about risk analysis, and we need to build a much better foundation in order to reliably carry out that function. This is where Evolutionary Project Management (Evo) methods are helpful. If you do not implement your designs evolutionarily, you might never learn that a particular one of them was your downfall. Even if you do learn which one it was, it is probably too late to do anything about it on this project.

If you are using Evo, a review could even deliberately decide to implement a high-risk step to find out the results. It is the benefits obtained that count, not the total avoidance of risk!

The key benefit of more frequent reviews is that risk is made more manageable. This might mean that lower level management can be empowered to make the review decisions. Alternatively, it could mean the design review itself might be come obsolete, since reality is going to give designers more reliable advice than any committee.

Conclusions

Any design specifications input into a design review must be of known high clarity and completeness: they should have successfully exited from Specification Quality Control (SQC) using both design specification rules, and then later using design specification review rules.

Design reviews should be held throughout the lifetime of a system. They should be held at early stages on samples of the design work to ensure initial problems and misunderstandings are detected as soon as possible. Reviews should also be held at appropriately frequent intervals, to avoid giving reviewers too much to review at one time.

For an evolutionary project, reviews should be held to decide/agree each 'next' evolutionary step. By utilising feedback, reviewers will learn more about their mistakes and successes, and any actions required to correct/improve project progress can be taken.

A design review should not be an informal management meeting to collect unfocused opinions under pressure.

References

Gilb, Tom and Dorothy Graham, *Software Inspection*, Addison-Wesley, ISBN 0-201-63181-4, 471 pages, 1993. (13th printing in 2005).

Gilb, Tom, *Competitive Engineering: A Handbook for Systems Engineering, Requirements Engineering,and Software Engineering using Planguage*, Elsevier 2005. Both SQC, Design and Evo have their own chapters in this book.

Haley, T., B. Ireland, Ed. Wojtaszek, D. Nash, R. Dion, *Raytheon Electronic Systems experience in Software Process Improvement*, Raytheon, 1995. This paper is available online at http://www.sei.cmu.edu/publications/documents/95.reports/95.tr.017.html

Johnson, Jim, Karen D. Boucher, Kyle Connors, and James Robinson, "Collaborating on Project Success," Software Magazine, February 2001. www.softwaremag.com/L.cfm?Doc=archive/2001feb/CollaborativeMgt.html

Morris, Peter W. G., *The Management of Projects*, London, Thomas Telford, ISBN 0-727-71693-X, 358 pages, 1994.

Taylor, Andrew, "IT projects sink or swim," BCS Review, 2001. http://www.bcs.org.uk/review/2001/articles/itservices/projects.htm

Biography

Tom has been an independent consultant, teacher and author, since 1960. He mainly works with multinational clients; helping improve their organizations, and their systems engineering methods.

Tom's latest book is 'Competitive Engineering: A Handbook For Systems Engineering, Requirements Engineering, and Software Engineering Using Planguage' (Summer 2005).

Other books are 'Software Inspection' (with Dorothy Graham, 1993), and 'Principles of Software Engineering Management' (1988). His 'Software Metrics' book (1976, OoP) has been cited as the initial foundation of what is now CMMI Level 4. Tom's key interests include business metrics, evolutionary delivery, and further development of his planning language, 'Planguage'. He is a member of INCOSE and is an active member of the Norwegian chapter NORSEC. He participates in the INCOSE Requirements Working Group, and the Risk Management Group.

www.diazhilterscheid.de

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

ISTQB®
International Software
Testing Qualifications Board

81,

ISTQB Certified

Are you on th

www.

* Reported March 2008

855*

Testers Worldwide

ne right track?

.istqb.org

Turning Software Testing into the Backbone of your Quality Assurance System

by Thierry Duchamp

How to Institutionalize Testing in Your Corporation

Synopsis:
This article describes an efficient way to create a global quality improvement programme based on an existing testing strategy. This approach drastically improves the quality of the software which will be apparent to customers or end-users. This approach uses a global quality system (e.g. CMM-I) that guarantees coherency of quick-wins in a longer term quality improvement programme.

What do you expect from testing?

Understand your context
Information Technology employees in the banking industry and more generally speaking in the software industry, often consider Quality Assurance (QA) as the equivalent of "testing". Furthermore, they often misinterpret testing as functional testing. Quality improvement models such as CMM-I give QA a different meaning: a department that decides, implements and controls best practices for delivering high-quality software. CMM-I defines testing as "Quality Control". This is sensible if you prefer an industrial approach to a non-industrial model based on very rare (and therefore expensive) technical and business experts.

An organization that wrongly considers testing as the equivalent to quality assurance is often an organization that addresses quality issues in a way that is not industrialized. This article is not about explaining the advantages of an industrialized model over a non-industrialized model. There are many profitable businesses that do not rely on mass-industry best practices. Choosing one or the other is a matter of scale (number of customers), competition (how good is the quality of their product), time to market (how fast new versions must be released), and there are of course other factors. The most important things for organizations are agility and flexibility, which allow them to adapt when their business grows and time to market gets crucial. I believe that testing is an important building block for moving towards a managed, agile and flexible approach.

Is testing really an efficient way to improve the quality of software?
The question might sound very provocative, especially if you invest a lot of money in testing. To be even more provocative, the answer is definitely "no", although of course testing does indirectly have positive effects. Then, why do corporations invest so much in testing? What is the purpose of spending up to 30% of the global budget of R&D in testing? From what I have seen in the software and banking industries, organizations start to put in place specific testing teams when the lack of quality impacts profits or production systems. Most of the time they put in place a team called "Quality Assurance", and stakeholders expect immediate and high return on their investment and a drastic increase of customer/end-user satisfaction. Senior managers expect to receive less escalations and complaints. This is how they can measure whether things are improving. In most cases, results are at first far below expectations. However, I believe that this is normal when implementing an industrialized quality improvement system.

Unfortunately, this initial step has many counter-productive and destructive effects such as:

- A decrease in the quality of the code, because developers feel a false sense of security and assume that the "Quality Assurance" net will catch every defect.
- Too much invested in testing instead of solving root causes for poor quality, such as requirement management or quality of fixes.

Furthermore, testing 100% of applications would make most businesses non-profitable and difficult to deliver on time. Since professionals cannot build this 100% bug-proof security net at a reasonable price, they have to answer three questions:

- How useful are the tests they run?
- How much should they invest to improve the software?
- Which tests give maximum return?

Let's consider testing as measurement (Quality Control)
If software is an industry, then we can certainly apply industrial (i.e. mass production) best practices in this area. How do professionals handle testing in manufacturing? Even if highly automated, factories still use manual control for checking the quality of their products. Investment for doing so is often high because of the large number of items to test. So, even in low-cost areas, it is wise to control only a subset of the manufactured items. I have always considered this situation very similar to what happens in our industry. Complexity is not based on the number of items we produce. In IT, complexity is:

- the number of lines of code
- management of dependencies between software components
- the number of combinations of data that the system can handle.

So, how efficient is it to test with these complexities? Experience shows that it is often

Visure
the requirements company

No destination in mind, labyrinth to test.

Be ahead of your product development.

IRQA® *is the corporate Requirements Engineering solution, that allows defining tests according to your requirements specification.*

Visure Solutions, The Requirements Company, help customers improve the resulting quality of their products and services streamlining their Requirements Management & Engineering processes.

Visure Solutions, is supported by the IRQA state-of-the-art features endorsed by worldwide leading companies, Requirements Management & Engineering specialized support and professional services and an experienced team on development processes.

www.visuresolutions.com info@visuresolutions.com

quite poor and often below end-users' and stakeholders' expectations. Even the most efficient testing strategies (highly automated and customer-centric) provide far too weak security net to catch all or even a higher proportion of defects. We must consider testing differently: is testing just a measurement of quality? I know that this question might surprise you, but it is something common for mass-production industries.

Testing and measurements

Coupling testing with efficient measurement

In the past, testing in the past was often not defined and not managed. Developers and "QA engineers" ran whatever tests they wished as long as the organization delivered on time with a satisfactory level of quality. As quality was often below expectations, the "Quality Assurance Manager" was urged to answer three questions:

- How much does the organization spend on testing?
- What is actually tested?
- What is the return on investment?

Measuring your investment

Most companies track their employees' activities on a daily or a weekly basis, which is an excellent way of knowing what your exact investment is on testing. For an accurate measurement, we must take into account that people other than testers also participate in the testing effort (support engineers, developers, product managers). You might be very surprised to find out that some teams perform testing activities for which they were not originally hired. Knowing these facts can help you to understand the roots-causes of poor quality. It is important that the measurement of investment is complete and accurate. This measurement includes the costs for effort, hardware, software and maintenance costs.

Measuring test coverage of applications

Since I have been working in quality assurance, customers and stakeholders have often inquired about the test coverage of the application. The key factor for measuring test coverage is to document the test cases. This requires the replacement of informal testing by a repeatable testing process. Low-end solutions for doing this rely on manual testing. They entails having test scenarios that are stored in text documents and test results which are stored in spreadsheets. High-end solutions rely on software vendor solutions which include information systems coupled with test robots. Such solutions are able to replay tests and check results automatically in the application.

Of course, the high-end solutions are much more efficient, event though they might be very expensive (I am not talking about the price of the test software licenses here, but rather the total cost for setting up the tools and putting in place a dedicated team). Automated tests are repeatable, since one can launch the same set of tests for each release and check results. The way in which test engineers define test scenarios is a good way to calculate the test coverage. Defining an application as a set of features is understood by everyone, so this is an excellent reference for developers, managers, architects and end-users. The basic unit of measurement is the feature (i.e. a specific function point):

$$\text{Coverage} = \frac{\text{Number of tested features of the application}}{\text{Total number of features of the application}}$$

It is obvious that one has to describe 100% of the features in the application to get a reliable measurement. I cannot say that this is the most accurate measurement, but I have experienced that it is sufficient to put in place and institutionalise in an 18-month testing project. The measurement of coverage can easily be improved at any time. Depending on your context, you can also use other basic function points for measurement. As a further example, you can measure *Coverage* by the number of lines of code covered by the test, in proportion to the total number of lines of code.

At this stage in the document we assume that we can fix any defect found by internal testing (i.e. all tests have been successfully passed when the software is released).

How to measure the efficiency of testing

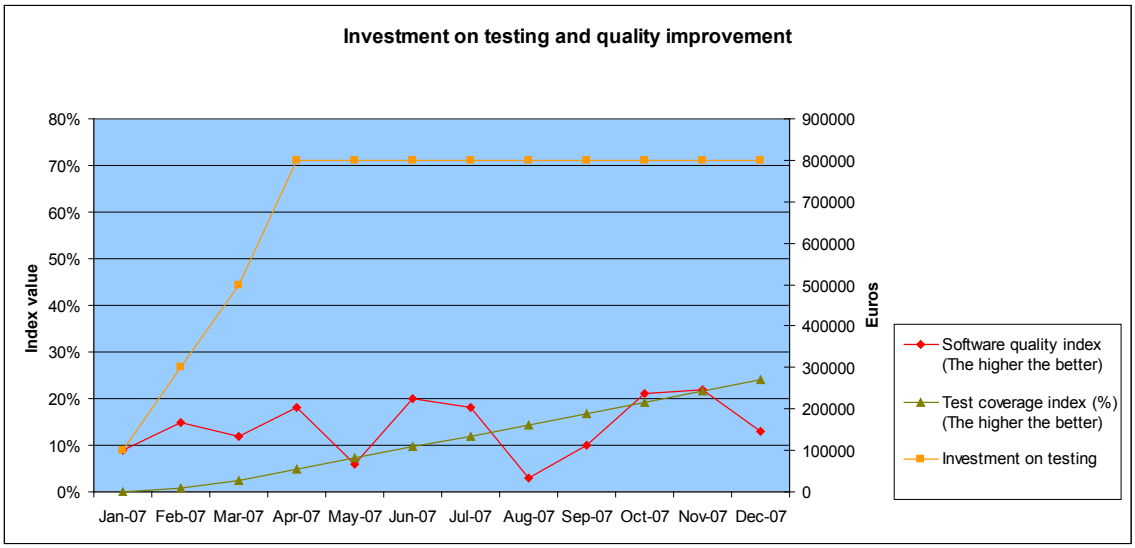
Test coverage is an internal measurement that must be coupled to a measurement of the quality of software as perceived by end-users. Support teams often have very useful information: service requests, escalations or interruption of services of production systems. These kinds of figures are more reliable and easier to use than customer satisfaction surveys that are more expensive to organize and often biased by external factors.

A typical measurement of quality is the sum of service requests raised by customers, weighted by severity:

```
QualityIndex (Month) = If (NonQualityIndex(Month) < 100)
                           100 - NonQualityIndex(Month)
                           Else 0

Where NonQualityIndex (Month) = 20 * #SR(sev1) +
                                5 * #SR(sev2) +
                                2 * #SR(sev3) +
                                    #SR(sev4)
```

Using these measurements proves that testing is not very efficient or not efficient at all: the *QualityIndex* is low. At this stage people in charge of putting in place the test strategy will need strong support from stakeholders and from all levels of management.



Example: Investment in testing directly increases test coverage but it is not efficient on the quality as perceived by customers/end users. Of course, there are other factors which impact the quality of the software (Please note that these figures are disguised)

Testing measures quality which is very effective to put in place corrective actions to develop better software.

How to leverage testing results

We can improve the things we measure. A quality improvement programme helps you to do the following:

- Invest the relevant amount of money for reaching a certain level of quality
- Invest in the right areas where you will get maximum quality improvement

I propose three steps to achieve this:

- Put testing in relation to the maturity of your organization thanks to a quality system
- Put in place corrective actions based on a gap analysis with industry's best practices
- Measure improvement with the help of metrics as described in the previous chapter

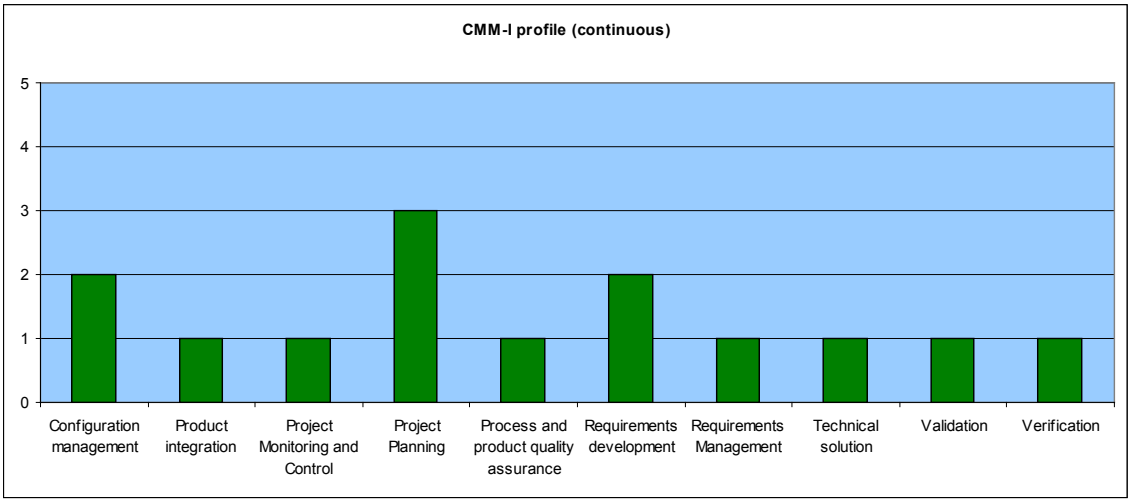
Put testing in relation to the maturity of your organization

Your organization has certainly implemented a quality system such as ITIL, CMM-I, COBIT or an internally approved methodology. If this is not the case it is always possible to quickly if only partially implement one of these methodologies for improving your testing strategy. You could not avoid hiring an expert who would produce a gap analysis document (which should take between 10 to 20 man days).

The gap analysis shows, among other things how your organization manages testing activities, which highly depends on:

- Your business model (software vendor, service, application service provider)
- Your business field (finance, telecommunication…)
- Other factors such as size and background of the organization

Below is the type of profile your consultant will supply. All examples are based on the CMM-I, continuous model).



Example of a CMMI profile (Please note that these figures are disguised)

We must now match this profile with the measurements described in the previous section (test efficiency and quality index).

Let's first focus on the areas (i.e. CMMI process areas) that are the closest to testing activities:

- Validation
- Verification

Here are the details for both process areas:

Validation										
SG 1 Prepare for Validation	1.1	1.2	1.3							
SG 2 Validate Product or Product Components	2.1	2.2								
GG2: Institutionalize a managed process	2.1	2.2	2.3	2.4	2.5	2.6	2.7	2.8	2.9	2.10

Example of result for PA Validation (Please note that these figures are disguised)

Verification										
SG 1 Prepare for Verification	1.1	1.2	1.3							
SG 2 Perform Peer Reviews	2.1	2.2	2.3							
SG 3 Verify Selected Work Products	3.1	3.2								
GG2: Institutionalize a managed process	2.1	2.2	2.3	2.4	2.5	2.6	2.7	2.8	2.9	2.10

Example of result for PA Verification (Please note that these figures are disguised)

We can guarantee the effectiveness of the quality improvement programme by ensuring:

- that it positively impacts the internal measurement test *Coverage*
- that test Coverage is a also a measurement of test efficiency and therefore is correlated to the *QualityIndex*

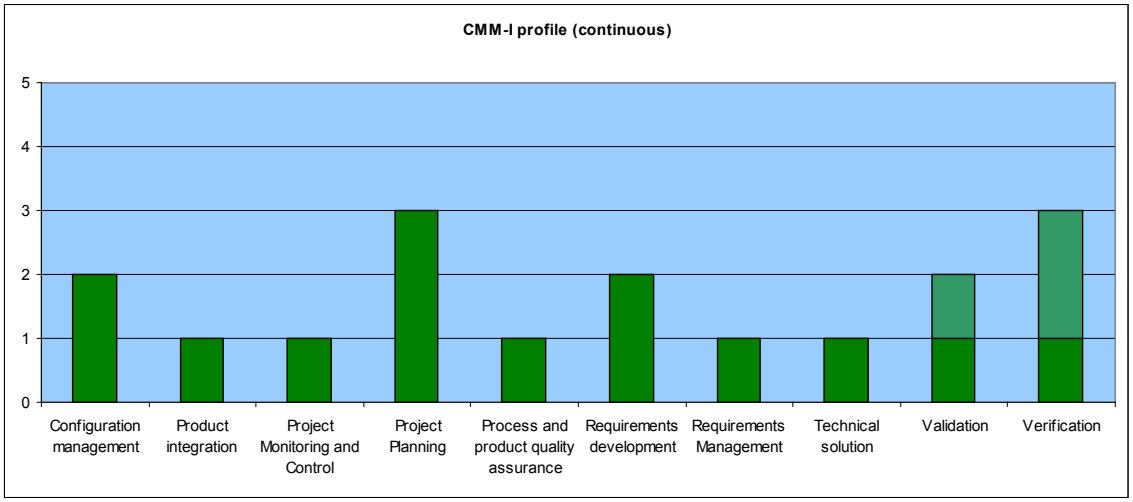
The CMM-I representation will help stakeholders to keep their focus on global improvement goals, while putting in place specific short-term actions which will achieve immediate results. You take the risk of many setbacks and failures if you decide to deploy corrective actions independently from a quality programme:

- It would be a set of independent short term actions which are often very confusing in the long run
- Actions could be contradictory
- Actions could address only specific areas which would not have a global impact the organization

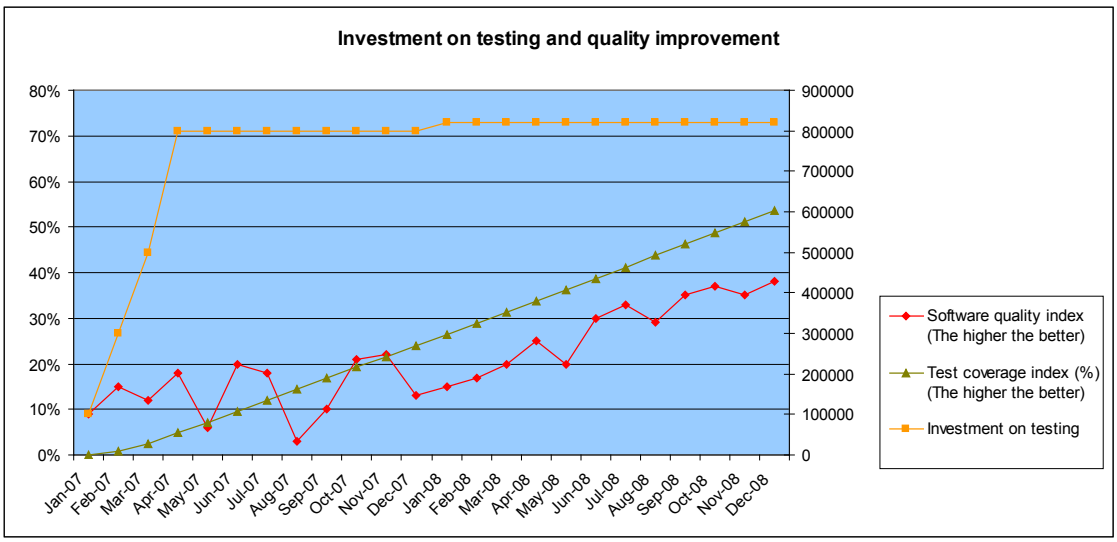
On the other hand, if the actions are part of a global quality improvement programme, efficient corrective actions will also:

- Improve results of any future appraisal
- Improve the quality of the software as perceived by customers and end users

If everything works as planned, you will be able to confirm that your testing strategy was successful. This will show that you have taken the proper corrective actions.



Example: Value of validation and verification increased after the deployment of the quality programme based on quick-wins (Please note that these figures are disguised)



Example: The quality improvement programme started in Jan-08, quick wins have direct effects. Extra investment is small compared to benefits (Please note that these figures are disguised)

Examples of corrective action and conclusion

I cannot give an exhaustive list of corrective actions that can be put into place, as this highly depends on your business and your organization. Corrective actions consist of deploying industry best practices that are non-existent or too poorly implemented:

- Peer review
- Standard coding guideline tools (static code analysis)
- Dynamic code analysis tools methodologies
- Functional testing
- Performance testing
- Formal beta testing
- Customer defects root analysis

You will certainly discover that what everyone calls testing/QA/QC in your organization will not cover all the entries of this list. Implementing new corrective actions from this list will allow you to get the best return for the lowest investment. Coupling testing to a quality programme is definitively the most efficient way to accelerate the improvement of quality at a reasonable investment level.

Biography

Thierry Duchamp is Head of Quality Assurance in a business unit of Reuters, which sells a financial risk management software suite. He has been working for the financial software industry for twelve years in various positions in development, testing and quality assurance.

Index Of Advertisers

ISTQB Certified Tester Training Worldwide - German, English, Spanish -

http://training.diazhilterscheid.com/

CONQUEST '08

11th international Conference
on Quality Engineering in Software Technology

Potsdam Chamber of Commerce and Industry, Germany
September 24 – 26, 2008

Register Now!

www.conquest-conference.org
info@isqi.org

25 % discount for Testing Experience readers

- 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

Keynotes

- **Libby Affen**, *Matrix Global Israel (Talpiot), Israel*
- **Andreas Kindt**, *T-Home, Germany*
- **Ingolf Krüger**, *University of California, San Diego, USA*
- **Andreas Spillner**, *Hochschule Bremen, Germany*

& **Karin Vosseberg**, *pdv.com Beratungs-GmbH, Germany*

Partner Country

ISRAEL

Organizer

iSQI International Software Quality Institute

Kindly supported by

LAND BRANDENBURG Ministerium für Wirtschaft

SeSamBB

Partners

IHK Potsdam

Fraunhofer FOKUS Institut für Offene Kommunikationssysteme

Requirements-Based Testing – Ambiguity Reviews

by Gary E. Mogyorodi, B.Math., M.B.A.

Introduction

This article provides an overview of the Requirements-Based Testing (RBT) process. RBT is a rigorous process for improving the quality of requirements and for deriving the minimum number of test cases to cover 100% of those requirements. RBT is comprised of two techniques: Ambiguity Reviews and Cause-Effect Graphing.

An Ambiguity Review is used in the requirements phase of software development to identify ambiguities in functional[1] requirements. The intent of an Ambiguity Review is to identify anything that is unclear, ambiguous or incomplete in the requirements. The elimination of these ambiguities improves the quality of those requirements.

Cause-Effect Graphing is a test case design technique that is performed once requirements have been reviewed for ambiguity, and after they have been reviewed for content. Requirements are reviewed for content to insure that they are correct and complete. The Cause-Effect Graphing technique derives the minimum number of test cases to cover 100% of the functional requirements to improve the quality of test coverage.

This article addresses the first RBT technique - the Ambiguity Review. A follow-on article will discuss the Cause-Effect Graphing technique.

So What is it about Requirements?

A requirements document describes **what** the behavior of a software system is expected to be. It is the starting point for all remaining phases of software development. Requirements should be written in a detailed manner so that there is no misunderstanding regarding what the expected system should do. I call this a testable requirement. Too often, not enough time is spent on the requirements document, so when the requirements are written, there is confusion as to:

1. what the stakeholders expect will be delivered
2. what the developers are expected to deliver
3. what the testers are expected to test

Therefore, it would benefit the entire project team if there is one clear, detailed, testable set of requirements that they can work from.

Why should we care about having Testable Requirements?

Testable requirements reduce development time by avoiding costly defects in later phases of the software development life cycle.

The Relative Cost to Fix an Error

The cost of fixing a software error is lowest in the requirements phase of the software development life cycle. This is because there are very few deliverables at the beginning of a project to correct if an error is found. As the project moves into subsequent phases of software development, the cost of fixing an error rises dramatically, since there are more deliverables affected by the correction of each error, such as a design document or the code itself. See Table 1 for the relative cost to fix an error at different phases in the development life cycle.

Table 1 – The Relative Cost to Fix an Error

Phase in Which Found	Cost Ratio
Requirements	1
Design	3-6
Coding	10
Unit/Integration Testing	15-40
System/ Acceptance Testing	30-70
Production	40-1000

(IBM, GTE, et. al)

The Distribution of Bugs

A study by James Martin showed that the root causes of 56% of all of the software bugs identified in projects are a result of errors introduced in the requirements phase. Of the bugs rooted in requirements, roughly half of them are due to poorly written, ambiguous, unclear and incorrect requirements. The remaining half of these bugs are due to requirements that were completely omitted. So there is plenty of room for improvement for writing clear, concise, unambiguous and complete requirements.

What's an Ambiguity Review?

An Ambiguity Review is a test of the requirements to ensure that they are written in a clear, concise and unambiguous manner. An Ambiguity Review occurs at the earliest phase of development, i.e. in the requirements phase. At

[1] Functional requirement - A functional requirement specifies what the system must be able to do, in terms that are meaningful to its users.

this point in a project, all the project team has is a description of what the system is intended to do. The Ambiguity Review occurs prior to the review of requirements for content by the domain experts. The intent of the Ambiguity Review is to provide the domain experts with a better-quality set of requirements to work from, so they can better identify missing requirements, and improve the content (completeness and accuracy) of all requirements. The Ambiguity Review is based on a checklist of 15 common problems people have writing requirements.

Let's Do an Ambiguity Review

An example is often a good way to illustrate a technique. I will use a requirements document entitled "The Postal Regulation" to illustrate the use of an Ambiguity Review. The draft version of the Postal Regulation requirements is provided in the box below.

The Postal Regulation (Draft)

The following postal regulation determines postage surcharges. This regulation applies only to parcels, not other types of postal items, and only to parcels sent to South America. For parcels which people mail to South America between December 1 and December 24 of each year, the system will apply these surcharges in addition to the standard postage:

Country & Weight Surcharge

Argentina	All weights	$11
Brazil	Weight > 33 pounds	$21
Brazil	< 33 pounds	$17

Figure 1 – The Original Requirements

The first step in an Ambiguity Review is to identify any words or phrases that are potentially ambiguous. I have highlighted a number of words that are ambiguities, and we will take a closer look at each one.

The Postal Regulation

The following postal regulation determines postage surcharges. This regulation applies only to parcels, not **other** types of postal items, and only to parcels sent to South America. For parcels which people mail to South America between December 1 and December 24 of each year, the system **will** apply **these** surcharges in addition to the **standard** postage:

Country & Weight Surcharge

Argentina	All weights	$11
Brazil	Weight > 33 pounds	$21
Brazil	< 33 pounds	$17

Figure 2 – The Original Requirements with Potential Ambiguities Identified

1. The word "other" in the phrase "other types of postal items" does not clearly define what types of postal items there are other than parcels, so we would want to get clarification as to what these other types of postal items are.

2. The word "will" identifies an ambiguity called a "Dangling Else". The sentence with "will" in it tells us what is normally expected to happen, i.e., that the Country & Weight Surcharge is applied for parcels that people mail to South America between December 1 and December 24. But the requirements do not tell us what happens for any other set of conditions. For example, the requirements do not tell us what the postage surcharges are if the parcels are mailed to South America but NOT between December 1 and December 24? So we have a dangling else.

3. The word "these" is a pronoun that makes a reference to what surcharges the system will apply. It would be clearer to make the reference explicit, such as the surcharges in Table 1, and eliminate the word "these".

4. The word "standard" does not clearly define the postage amount prior to the application of postage surcharges being applied. The author of the requirements probably understands what standard postage means. The sentence is written as if the author assumes that the reader knows what standard postage means, which may not be true.

The next step in the Ambiguity Review is to review the requirements and identify anything else that is ambiguous. From reviewing the requirements, the following additional Ambiguity Review questions can be asked:

- Is there such a thing as non-standard postage?
- What postage surcharges are applied to parcels mailed to South America between December 1 and December 24 going to Brazil that weigh exactly 33 pounds?
- What postage surcharges are applied to South American countries other than Argentina or Brazil?
- What currency are the postage surcharges in?
- What is the postage surcharge for postal items other than parcels?

After getting clarification from the domain experts on these issues, the ambiguities are eliminated and the requirements are revised (see Figure 3) to look like this (new text in red letters):

The Postal Regulation **for South America** (Version 1)

The following postal regulation determines postage surcharges. This regulation applies only to parcels, not other types of postal items, and only to parcels sent to South America. **Other postal items are envelopes and post cards. There is no postage surcharge for envelopes and post cards.** For parcels which people mail to South America between December 1 and December 24 of each year, the system will apply **the** surcharges **in Table 1** in addition to the standard postage of **$5.00 US**:

Table 1 - Country & Weight Surcharge between **December 1 and December 24**

Argentina	All weights	$11 **US**
Brazil	Weight > 33 pounds	$21 **US**
Brazil	**Weight = 33 pounds**	**$19 US**
Brazil	Weight < 33 pounds	$17 US
Columbia, Ecuador, Peru, Bolivia, Chile, French Guiana, Guyana, Paraguay, Suriname, Uruguay, Venezuela, and Falkland Islands	**All weights**	**$15 US**

For parcels which people mail to South America outside of December 1 to December 24 of each year, the system will apply the surcharges in Table 2 in addition to the standard postage of $5.00 US:

Table 2 - Country & Weight Surcharge outside the dates December 1 to December 24

(In a real life problem, the specific details of Table 2 would be supplied here. For this example, they have been excluded.)

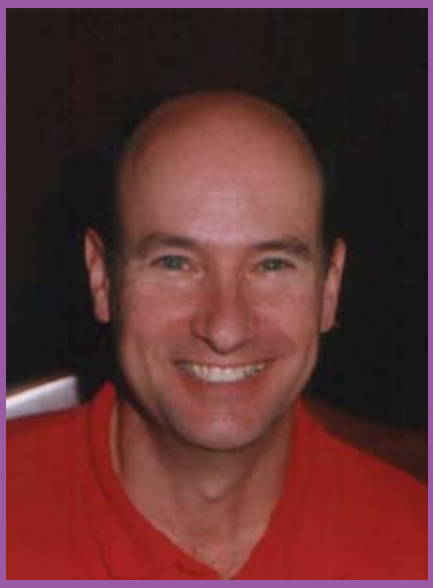
Figure 3 – The Postal Regulation Requirements Corrected for Ambiguity

In Summary

The goal is to derive a set of requirements that are testable. The requirement is testable if it is written in such a way that each statement is deterministic. Deterministic means that for a given starting condition and a set of inputs, the reader can determine exactly what the expected outcomes will be. Each statement in the requirements can then be used to prove or disprove whether the behavior of the software is correct. Eliminating ambiguities is the first step in deriving a testable requirement.

Once the requirements have been reviewed and corrected for ambiguity, they can be reviewed by the domain experts to ensure that they are correct and complete. Then the requirements are corrected for any errors or omissions.

Now the development team can move into the Design Phase of the software development life cycle with a complete set of requirements. At the same time, the testing team can begin test case design. In my next article, I will pick up from where we left off and show the use of the Cause-Effect Graphing technique to derive the minimum number of test cases that cover 100% of the functionality defined in the requirements for the Postal Regulation (for South America) problem.



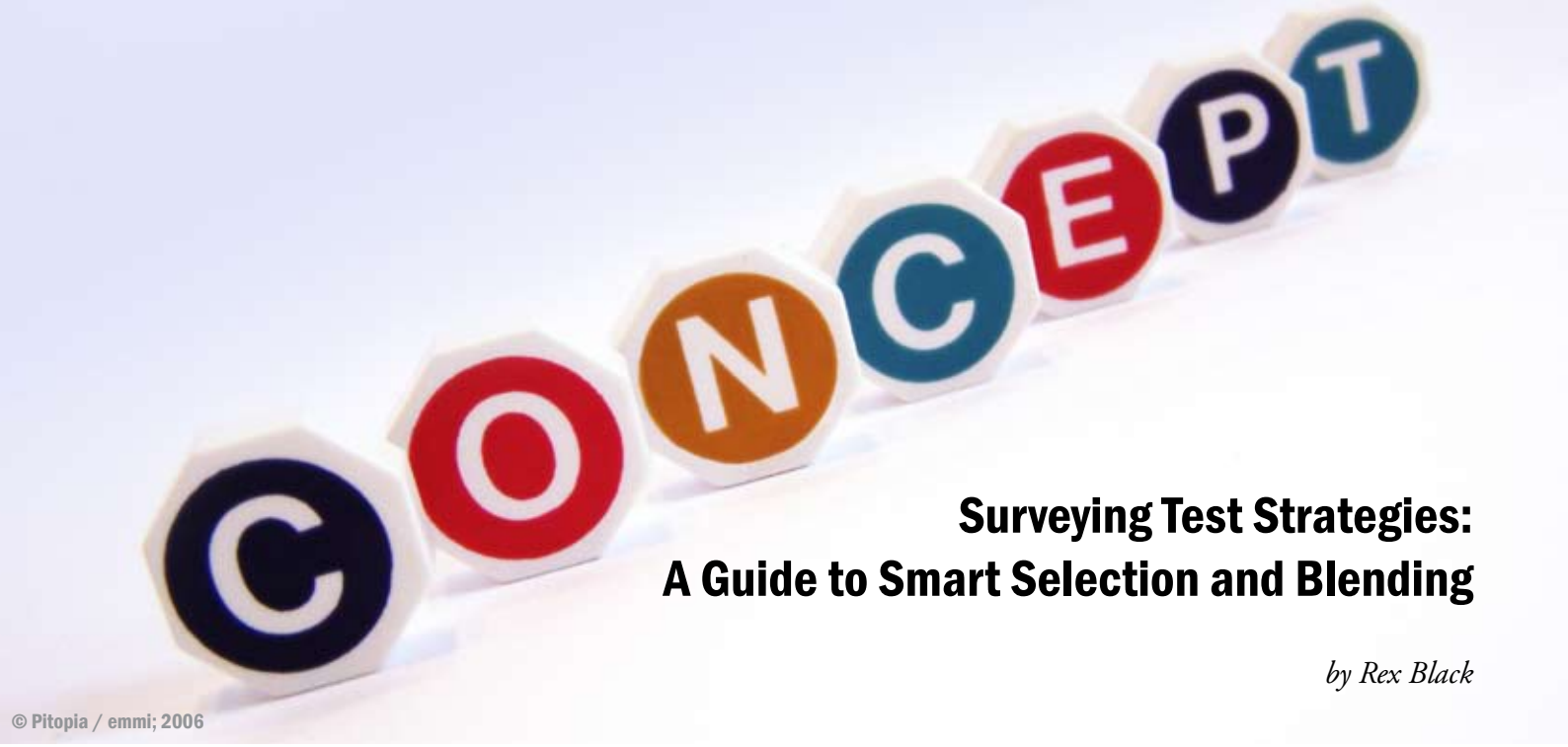
Biography

Gary Mogyorodi has over 30 years of experience in the computing industry. Currently, as Principal Consultant with Software Testing Services, he consults, trains and mentors in software testing, specializing in Requirements-Based Testing (RBT), i.e. Cause-Effect Graphing and Ambiguity Reviews. Some of his customers include CGI, FiLogix, H&R Block, Home Hardware, IBM, Manulife Financial, RBC, RBC Dexia, Siemens and TELUS. Gary Mogyorodi began practicing RBT with Bender & Associates in 1998. His career began at Dofasco Inc. in 1973. From 1993, he was a Quality Assurance and Software Testing specialist, managing testing efforts, developing testing methodologies, and creating standards and procedures for quality assurance and testing. Prior to that, he was as a Programmer, Systems Analyst and Manager of Software Development.
A prolific speaker, he has delivered presentations at the following engagements:

- CIPS (Canadian Information Processing Society), Toronto Chapter
- CIPS Hamilton Chapter
- TassQ (Toronto Association for System and Software Quality)
- CQAA (Chicago Quality Assurance Association)
- STAREAST and STARWEST Conferences
- Software Quality Forum
- Toronto SPIN (Software Process Improvement Network)
- Systems and Software Technology Conference
- PSQT/PSTT North and South Conferences
- SWOSQG (South Western Ontario Software Quality Group)
- International Quality Conference

Mr. Mogyorodi is the President of the Canadian Software Testing Board. He is an approved ISTQB Instructor at the Foundation Level.


We train in Latin America -
ISTQB Certified Tester Training
contact@bicorp.biz
© iStockphoto



Surveying Test Strategies: A Guide to Smart Selection and Blending

by Rex Black

Introduction

A test strategy provides a set of organizing principles for testing, as well as a project-independent methodology for testing. In some cases, the test strategy is documented. However, in this article, I won't examine how to document test strategies, but rather start at the beginning by discussing types of test strategies.

As test managers, we sometimes forget to think critically about why we do testing the way we do, and what our approach to testing does – and doesn't – offer us, i.e. the project team and the wider organization. Reading this article will help you analyze and improve your current test strategies as well as adding some new strategies to your toolbox.

There are many types of test strategies. I'll survey the most common ones. All of these strategies are in use by various test teams. Depending on the alignment of the strategies with the test mission and the project itself, any of these strategies can succeed – or fail.

Analytical Test Strategies

Analytical test strategies start with analysis as a foundation.

With an *object-guided* strategy, you look at requirements, design, and implementation objects to determine testing focus. These objects can include requirement specifications, design specifications, UML diagrams, use cases, source code, database schemas, and entity-relationship diagrams. As you might guess, this approach relies on extensive documentation, and breaks down when documentation is not available.

With a *risk-based* strategy, you use the informal or formal techniques discussed in the chapter on quality risk analysis to assess and prioritize quality risks. You can use various available sources of information, as with the object-guided strategy, but you should also draw on the insights of cross-functional project team members and other stakeholders. Adjust the tests and the extent of testing according to the risk priority levels. Unlike the object-guided variant, this approach can handle situations where there is little or no project documentation.

With a *fully-informed* strategy, you start with the object-guided or risk-based strategy, but take the analysis further. Study the system, the usage profiles, the target configurations, and as much other data as you can find. Design, develop, and execute tests based on the broad, deep knowledge gained in analysis. This approach is great if you have lots of time to research the system.

The items being analyzed are sometimes called the *test basis*. The results of the analysis guide the entire test effort, often through some form of coverage analysis during test design, development, execution, and results reporting. These strategies tend to be thorough, good at mitigating quality risks and finding bugs. However, they do require an up-front investment of time.

Model-Based Test Strategies

Model-based test strategies develop models for how the system should behave or work.

With a *scenario-based* strategy, you test according to real-world scenarios. These should span the system's functionality. In the object-oriented world, a close relative is the *use-case-based* strategy, where you rely on object-oriented design documents known as use cases. These use-cases are models of how users, customers, and other stakeholders use the system and how it should work under those conditions. You can translate these use-cases into test cases.

With a *domain-based* strategy, you analyze different domains of input data accepted by the system, data processing performed by the system, and output data delivered by the system. (Domains are classifications based on similarities identified in inputs, processing, or outputs.) Based on these domains, you then pick the best test cases in each domain, determined by likelihood of bugs, prevalent usage, deployed environments, or all three.

With a model-based strategy, you design, develop, and execute tests to cover models you have built. To the extent that the model captures the essential aspects of the system, these strategies are useful. Of course, these strategies rely on the ability of the tester to develop good models. These strategies break down when the models cannot – or the tester does not – capture all of the essential or potentially problematic aspects of the system.

Methodical Test Strategies

Methodical test strategies rely on some relatively informal but orderly and predictable approach to figure out where to test.

With a *learning-based* strategy, you use checklists that you develop over time to guide your testing. You develop these checklists based on where you've found (or missed) bugs before, good ideas you've learned from others, or any other source.

With a *function-based* strategy, you identify and then test each and every function of the system, often one at a time. Similarly, with a state-based strategy, you identify and test every state and every possible state-transition that can occur.

With a *quality-based* strategy, you use a quality hierarchy like ISO-9126 to identify and test the important "-ilities" for your system. For example, some groups in Hewlett-Packard use functionality, localization, usability, reliability, performance, and scalability. IBM uses capability, usability, performance, reliability, installability, maintainability, documentation, and operability.

With a methodical test strategy, you follow these standard inventories of test objectives. These strategies can be quick and effective for systems that remain relatively stable or systems which are similar to those tested before. Significant changes might render these

strategies temporarily ineffective until you can adjust the test objectives to the new system or organizational realities.

Process-Oriented Test Strategies

Process-oriented test strategies take the methodical approach one step further by regulating the test process.

With a *standardized* test strategy, you follow official or recognized standards. For example, the IEEE 829 standard for test documentation, created by a volunteer standards committee of the non-profit Institute for Electronics and Electrical Engineers, is used by some organizations to ensure regularity and completeness of all test documents. My book, *Critical Testing Processes*, describes twelve comprehensive, customizable, and lightweight processes for testing. Such standardization can help to make the test process transparent and comprehensible to programmers, managers, business analysts, and other non-testers. However, you must take care not to introduce excessive, wasteful, or obstructive levels of bureaucracy or paperwork.

One increasingly-popular test strategy, the *agile* test strategy, has arisen from the programming side of software engineering. Here, the testing follows lightweight processes, mostly focused on technical risk (likely bugs). A heavy emphasis is placed on automated unit testing, customer acceptance testing, and being able to respond to late changes without excessive costs. These strategies are tailored for small teams on short projects with immediate access to the users. Large, long, geographically distributed, or high-risk projects are likely to find that the strategy does not scale.

The topic of automated unit testing brings us to a group of test strategies that rely heavily on automation. One such strategy is an *automated random* test strategy, where a large amount of random input data are sent to the system. Another such strategy is an *automated functional* test strategy, where you test system functionality using repeatable scripts. Either strategy might also involve an automated load, performance, or reliability testing element. These strategies rely on the ability to effectively automate most of the testing that needs to be done.

Dynamic Test Strategies

Dynamic test strategies, like the agile test strategies, minimize up-front planning and test design, focusing on making the test execution period responsive to change and able to find as many bugs as possible.

With an *intuitive* test strategy, you test according to the collective experience, wisdom, and gut instincts of the test team. Discussions about what to test, anecdotal evidence from past projects, and oral tradition are prime drivers. With an *exploratory* test strategy, you simultaneously learn about the system's behavior and design while you run tests and find bugs. You continuously refine the test approach based on your test results, and re-focus the further testing. With a *bug hunting* strategy, you use bug profiles, taxonomies (classifications), and hunches (bug assumptions) to focus testing where you think the bugs are.

The hunting metaphor is a good one for all these strategies, which are more alike than different. I hunt and fish. I've learned one critical success factor to both sports: Hunt where the birds are and fish where the fish are. Likewise, these test strategies require that you be right about where you think the bugs are, often under conditions of schedule and personal pressure.

Dynamic test strategies value flexibility and finding bugs highly. They do not usually produce good information about coverage, systematically mitigate risks, or offer the opportunity to detect bugs early in the development lifecycle. They are certainly much better than no testing at all and, when blended with analytical strategies, serve as an excellent check-and-balance that can catch gaps in the analytically designed tests.

Philosophical Test Strategies

Philosophical test strategies start with a philosophy or belief about testing.

With an *exhaustive* test strategy, you assume that everything and anything can and will have bugs. You decide that the possibility of missing bugs is unacceptable, and that management will support a considerable effort to find all the bugs. You attempt to test extensively across the functionality, the quality risks, the requirements, and whatever else you can find to cover. The essence of this strategy is captured in an old tester's joke, derived from the catchphrase on the back of US currency: "In God we trust…all others we test."

With a *shotgun* test strategy, you also assume that everything and anything can and will be buggy. However, you accept that you cannot test everything. Since you lack any solid idea on where to find bugs, you test wherever and whatever comes to mind. You attempt to randomly distribute the test effort within the given resource and schedule boundaries, like pellets from a shotgun.

With an *externally-guided* test strategy, you accept that you cannot test everything, nor can you know where the bugs are. However, you trust that other people might have a good idea of where the bugs are. You ask for their guidance. You test according to their direction, including asking them to help you decide if the observed results are correct. Common guides include programmers, users, technical or customer support, help desk staff, business analysts, sales people, and marketing staff.

If the underlying philosophies and beliefs behind these strategies are correct, they can be appropriate. For example, testing weapons systems like nuclear missile guidance software clearly requires an exhaustive strategy. However, when applied in inappropriate situations—and I'd guess that most projects are inappropriate for at least two if not all three of these strategies—they lead to dangerously misaligned test efforts.

Regression and Regression Risk Strategies

Regression is the misbehavior of a previously-correct function, attribute, or feature. The word is also used to mean the discovery of a previously-undiscovered bug while running a previously-run test. Regression is generally associated with some change to the system, such as adding a feature or fixing a bug.

Regression falls into three basic types. The first is a local regression, where the change or bug fix creates a new bug. The second is an exposed regression, where the change or bug fix reveals an existing bug. The third is a remote regression, where a change or bug fix in one area breaks something in another area of the system. Of the three, the last is typically the hardest to detect, but any one of these regressions can slip past us if we're not careful.

Regression can affect new features. In Figure 1, you see a new development project that will produce n new features. The features are built, integrated, and tested one after another or in increments. Feature 3, it turns out, breaks something in feature 1.

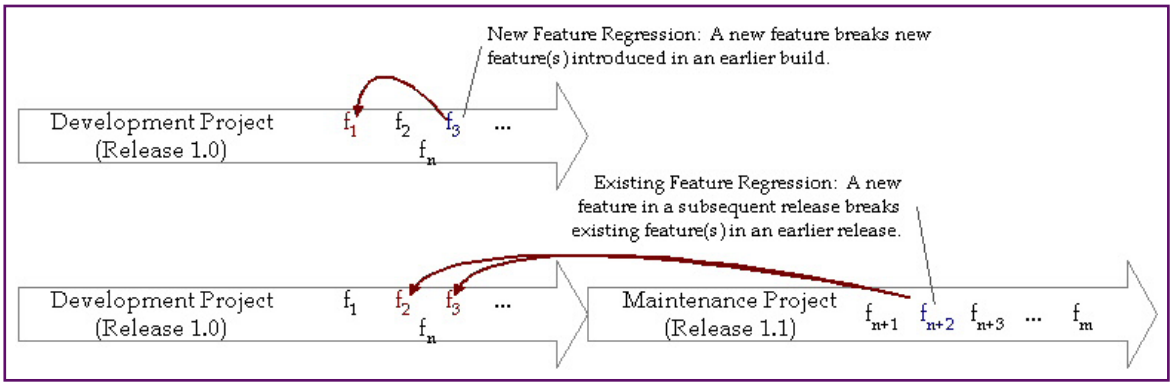


Figure 1: Types of regression

Regression can affect existing features. In the figure, you see a maintenance project that will add m new features in release 1.1. The features are added on top of the existing n features that were present in release 1.0. Feature n+2 breaks existing features 2 and 3.

Of the two effects, breaking existing features is typically worse. Users, customers, and other stakeholders come to rely on existing features in a system. When those features stop working, so do they. For new features, a user, customer, or other stakeholder might be disappointed not to receive the promised capability, but at least it was not a capability around which their daily work has become entwined.

So, what test strategies exist for dealing with regression?

Regression Risk Strategy 1: Repeat All Tests

For regression risk mitigation, a brute force strategy is to repeat all of our tests. Suppose you've developed a set of tests that are well aligned with quality. You'll have done that if you've performed a solid quality risk analysis and received sufficient time and resources to cover all the critical quality risks. If we repeat all tests after the very last change to the code, we should find all the important regression bugs.

Realistically, automation is the only means to repeat all tests for large, complex systems. Automation is practical when you can design and implement automated tests where the high up-front test development costs are recouped, through frequent test repetition, by the low (ideally, close to zero) costs of executing and maintaining the tests. This test automation can occur at a graphical user interface (GUI), at an application program interface (API), class function or method interface, at a service interface like those in a networked system with a service-oriented architecture, or at the command line interface (CLI).

Regression Risk Strategy 2: Repeat Some Tests

For one reason or another, it is often not possible to repeat all tests. So, you must select some subset of your overall set of tests to repeat. There are three major techniques for doing so.

The first is the use of traceability. Briefly, traceability is where tests are related to behavioral descriptions of the system like requirement specification elements, design specification elements, or quality risks. We can look at what requirement, design element, or quality risk is affected by the fix or change, trace back to the associated tests, and select those tests for re-execution.

The second is the use of change analysis. In this case, you look at structural descriptions of the system to figure out how change effects could ripple through the system. Unless you have an in-depth understanding of programming and the system's design, you'll probably need help from programmers and designers.

The third is the use of quality risk analysis. With traceability and change analysis, you are using technical risk—where bugs are likely to be found—to decide what to retest. However, you should also revisit your quality risk analysis to determine what areas have high business risk. Even if bugs are unlikely, areas with high business risk should be retested.

Since you're not rerunning every test, you are likely to miss regressions if they occur in unanticipated areas. So, you can use cross-functional tests to get a lot of accidental regression testing.

The analogy is clearing a minefield. The old-fashioned approach of clearing minefields by walking through the field looking for each mine one at a time has been superceded by the use of large earthmoving equipment fitted with heavy rollers and flails. These clear huge swaths of the minefield at once, and are unlikely to miss anything.

Likewise, a cross-functional test touches lots of areas of the system at one time. The key assumption behind cross-functional tests is that you won't run into too many bugs. If you do, then you'll not get through any single test, as you'll get blocked partway into your cross-functional voyage through the system.

I have a client that builds a large, complex, high-risk geological modeling package for industrial use. They have a test set that is so large and complex that it takes them a whole year—the length of their system test period—to run each test once, with only a small percentage of tests re-executed to confirm bug fixes. They rely on cross-functional tests to help mitigate their regression risk.

They also use code coverage to help them assess the level of regression risk. In this case, the idea of code coverage is that the greater the percentage of the program you have tested, the less likely it is that you'll miss bugs. Code coverage measurements allowed my client to make sure that 100% of the statements in the program had been tested at least once in the one-year test period, which mitigated the risk of breaking existing features. Code coverage measurements also told them that, thanks to cross-functional testing, in any given four-week period, they had executed between 50 and 70% of the statements in the program. This not only mitigated the risk of breaking existing features, but also the risk of breaking new features.

Conclusion

At this point, we've surveyed the primary types of test strategies in use in most organizations for new systems. Most testers and test managers will recognize their own approach in the list, as well as learning a few new types. In some cases, when we've reviewed these types for clients and course attendees, they start to question why they use their current approach, given its strengths and weaknesses. That's good, because, as I mentioned at the start, thinking critically about why we do what we do is essential to achieving an effective, efficient testing process.

As you think about how to improve your test strategy, remember that these strategies in the real world can be—and should be—adapted to their context. In this article, what I've described are pure forms, as you might refer to shapes as triangles, circles, squares, rectangles, and so forth. Of course, objects in the real world include features of many types of shapes. The point is not to achieve test strategy purity, but rather a test strategy that is fit for purpose in your testing.

In addition, remember that strategies are not philosophies or religions—you don't have to choose just one and stay with it. Strategies may be combined. You don't have to choose a single strategy. While my dominant strategy in most cases is a risk-based analytical strategy, I often blend and use multiple strategies.

Finally, be sure to spend some time on carefully selecting and tailoring your test strategies, not just once, but for each project. You can adopt general strategies for your test team, but remember to be ready to modify or completely re-work those for a particular project, too.

Best of luck to you as you take a more strategic approach to your testing!

Acknowledgements

This article is an excerpt from Rex Black's book, Pragmatic Software Testing, available from Wiley. The taxonomy of test strategies presented here grew from an online discussion between Rex Black, Kathy Iberle, Ross Collard, and Cem Kaner, and Rex thanks them for stimulating the thoughts that lead to this list.

Biography

With a quarter-century of software and systems engineering experience, Rex Black is President of RBCS (www.rbcs-us.com), a leader in software, hardware, and systems testing. For over a dozen years, RBCS has delivered services in consulting, outsourcing and training for software and hardware testing. Rex is the author of four books on software testing that have sold some 50,000 copies. He has written over twenty-five articles, presented hundreds of papers, workshops, and seminars, and given about thirty keynote speeches at conferences and events around the world.

www.bicorp.biz

Development & Testing Outsourcing in Latin America

Business Innovations

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

Finally Usability Testing?

State-of-the-practice

I recently came upon some case study papers regarding usability testing of web site. I assume that this means (some) attention is provided to one of the most critical success factors for websites: usability and thus usability testing. In fact usability has been identified in various surveys as one of the main reason for project success in any IT project! For a number of years I've been lecturing on this topic at test conferences. At EuroSTAR, I have received the best tutorial award for "Discount Usability Testing" and our company Improve Quality Services regularly runs courses on "Usability Testing". Yet, I don't have the impression that all this pioneering work has had much impact on the testing community. During the latest EuroSTAR conference an usability expert from South Africa stated that "usability (testing) will only start to be performed and receive the right level of attention, when testing for functionality is under control". An interesting thought that, as far as I'm concerned, is very much true. Perhaps we were just not ready for it, are we now…..?

Best practices

It remains strange that when no less than 40% of the software code that is being developed world-wide is directly or indirectly related to the user-interface, we still only dedicate a minimum percentage of our test effort to usability testing. Yet, there are many so-called best practices available. Only last week I was involved in a usability test that very clearly showed that thorough attention for usability throughout the development process (requirements, standards, expert reviews, usability testing) can deliver great results. The users were enthusiastic regarding the new system that supported their tasks in an efficient, effective and user-friendly manner. This is also a possible in real-life IT-project!

Practical recommendations

As far as I'm concerned every (senior) tester that operates at system level should, in additional to the conventional functional test design techniques (boundary value analysis, equivalence partitioning, decision tables etc.), also have the level of knowledge and skills that allows them to apply a number of relatively easy to use usability test techniques. The first things that come to my mind in this context are the Heuristic Evaluation technique developed by Jacob Nielsen (www.useit.com) and the SUMI questionnaire method (www.improveqs.nl). Both techniques are relatively easy to learn, do not require more than one of two days of testing effort and therefore have a great cost/benefit ratio. Recently the British Computer Society (BCS) Special Interest Group in Software Testing developed a testing standard in the area of usability testing that is certainly worthwhile to get acquainted with (www.testingstandards.co.uk). Sufficient possibilities therefore to make a good, low-costs but thorough start with usability testing; preferably in co-operation with developers, prevention is always better than cure.

During a recent conference a number of enthusiastic former participants to my usability courses told me they were now using the techniques mentioned above in their test projects with good results. Usability testing, will it finally happen?

For queries or comments you can contact Erik van Veenendaal (eve@improveqs.nl)

Tea testers look at aspects of dried & brewed tea leaves and the liquor

Likewise, PureTesting goes into various aspects & nuances of software testing

We build innovative, end-to-end solutions,
and manage critical testing processes,
and reduce total cost of producing quality software

- Banking & Finacial Services
- Pharmaceuticals
- eLearning
- Datacom & Telecom
- Embedded Systems
- EAI & ISVs

Test Consulting • Testing Services • Testing Products

PureTesting

Testing Thought Leadership

India • USA • UK • NZ

www.puretesting.com

+91 (120) 4621010; info@puretesting.com

Global Software Test Consulting & Services company

Requirements and test management: More safety in vehicles through traceable testing

by Norbert Ruck, Ralf Lulei, Thorsten Geiselhart and Gerd Rockweiler

Testing is generally understood as concerned with finding defects in systems under development. The objective is to gain confidence in knowing whether a technical development functions as expected and whether it behaves in a consistent manner. In this sense the test is not meant to be a guarantee for absolute correctness. Instead testing makes it possible to determine the circumstances and the likelihood that deviations from expected outcomes will occur. In any case testing should reduce the number of defects in the system be able to demonstrate as far as possible that the system functionality is implemented.

Automotive sector

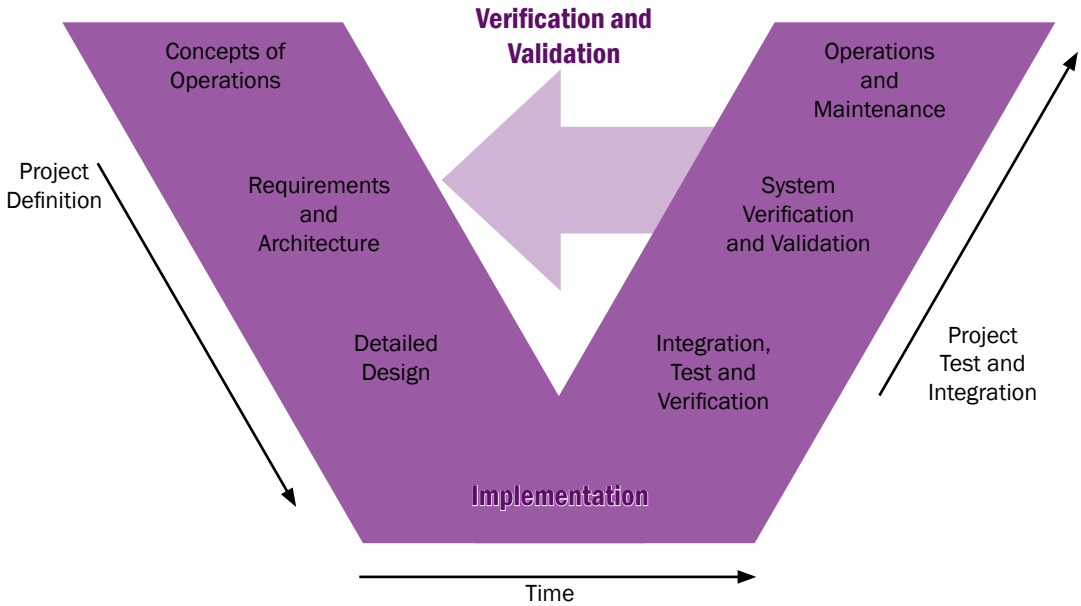
In many sectors failures can be handled with low priority because the end user doesn't notice them or because the system usability does not suffer. When people can be endangered by such failures, safety takes on a different significance. This is why testing is given a high level of significance in the automotive sector. This is accommodated by international standards and to no small degree by internal company processes. These regulate in detail the path from customer requirements to test cases so that a complete and documented process can be established upon which a car's control unit can be developed which poses absolutely no risk to its occupants.

The so-called V-model has proven its worth here by providing a sequence with increasing levels of detail for the development of software projects, which we also find used in the automotive sector (see diagram).

An automobile manufacturer fundamentally lays down the requirements on a software system in a customer requirements specification document (in general, for an automobile's control unit). The functional specification then sets out how these requirements will be implemented. This is the basis for the creation of test cases which are linked to the functional specification and thereby enable evidence to be provided that test cases exist for all requirements. Reviews verify the correctness and completeness of each step.

Process-driven engineering

Within the life-cycle of a product a series of documents are to be produced, examined and reviewed so they can ultimately be corrected or extended. In many cases a standard MS-Office solution is employed which is in any case already in company use. This probably means that the customer requirements and functional specification are created with a word processor whereas the test case specifications and test protocols are mostly stored in a spreadsheet. Even though this practice benefits from low procurement and learning costs, the disadvantage is that the compliance with required standards and process requirements are made more difficult.



This is particularly noticeable when reacting to requirements changes, to ever faster software release cycles and when showing that all requirements are covered by test cases. For these purposes the use of a special software solution is recommended which replicates the development process. Marquardt GmbH, a company which amongst other things produces software for the

MÉTODOS Y TECNOLOGÍA

The leading TTCN company

www.mtp.es

For further information call us:
(+34) 91 353 15 64.

TTCN Testing **SERVICES**

- Test Suite Design and Implementation.
- Test Environment Solutions (Codecs, Adapters...).

Testing **TURNKEY PROJECTS**

TTCN Certified **TRAINING**

ACCREDITED ISTQB

Powerful Test Automation TTCN-3 **TOOL**

ttcn EXHAUSTIF

OUTSOURCING Capabilities: Spain, Brasil, Uruguay, Mexico.

Engineering and Software Quality Consultancy

METODOS Y TECNOLOGIA • Paseo de la Castellana 182, 10° planta • 28046 Madrid/SPAIN • Tel.: +34 91 353 15 64

control units used by several large automobile makers, employs the IRqA (Integral Requisite Analyzer Requirement tool) from Visure Resolutions. The objective here is to achieve the customer's requirements as quickly as possible with maximum quality while at the same time demonstrating full implementation and ensuring traceability across all documents. And this has to be within the entire product life-cycle. We will now show the implementation of this process at Marquardt GmbH.

Requirements management tool

Within the tool itself there are separate areas called "Engineering Processes", which include the management of actual requirements and the management of test scenarios. Various roles which are implemented in the tool permit individual user rights to be allocated for these areas. In this way the software developer receives read-only rights for the „Test Scenario Process" and at the same time has write privileges for the „Requirements Management Process". For the software tester the opposite applies. The separate areas within the requirements management tool make such a role-specific separation possible and appropriate, all required data can be viewed but only role-specific information can be modified.

A further advantage which results from the use of a common database is traceability. This means the linking from customer requirements (in the requirements document) to the functional specification (in the functional requirements document) through to validation that a complete coverage of all requirements in the system tests can be shown. This is presented in a so-called traceability matrix, which shows which requirements are tested by which test cases and where possible holes in the test coverage are present. A further major advantage arises when a change is made on the requirements side, irrespective of whether the requirements document or the functional specification is changed. These changes show up as "suspected links" in the traceability matrix. These show at a glance which requirements have changed and for which test cases modifications will be needed. In this manner the searching and location through various versions of the documentation history becomes redundant. Everything is shown in the central requirement management tool.

The data base within IrqA permits all users to access current data at the same time. A check-out mechanism ensures that only a single specific user can change a particular data record at a given time. In this case other users can instantly read the data without requiring that the author first check the data record in. In this way any dependent data records which may exist can be immediately modified. In addition, further users have the possibility to work with other information. This allows work to take place on various data records in parallel and leads to significant synergy effects.

The history function allows earlier versions of data records to be compared with the current version. At the end of each development cycle a so-called baseline can be set for all data records or for partial areas. This represents a "freeze" of the captured data, which later allows access to the exact data record that was current, reviewed and released in this software release.

Engineering process - Test scenario

The administration of test requirements at Marquardt GmbH takes place in a five stage hierarchical structure. This is constructed as follows: test root - test scenario – test object – test group – test case.

Test root is the basis for all data records within the IRqA engineering process. All test scenarios are summarized in the test root. According to this nomenclature a test scenario contains all test objects. In this sense the system test, for example, is represented as a test scenario. According to the scale of the project the tests of individual system components are also test scenarios. For complex projects with many components there are several test scenarios and for simple projects with just one component a single test scenario is sufficient.
Each test scenario is divided into several test objects. The test objects are divided up into the functional aspects of the item under test (e.g. CAN, LIN, operational voltage) and the test objects divided up into test groups within each of these functional aspects. This results in easier administration of the test cases. Test objects within the LIN test scenarios can, for example, be Wake Up, LIN-failure or communications error.
The test cases themselves are at the lowest level in the hierarchy. This consists of a unique code, the link to the appropriate requirement and a name from which the test content can be deduced. A list of codes and names can be made available as a high level specification at the request of the customer. The actual test case content and procedure is represented by descriptions of preconditions, actions, results and remarks. Furthermore, each test case is classified according to risk, and with the help of an additional attribute the distinction between positive and negative test case can be made. Further attributes are available for managing the test cases.
The block structure is administered hierarchically within the engineering process. In this manner the test root, the test scenarios and the test objects each receive their own block, whose attributes are inherited from the test root. This guarantees a consistent management of test cases across all blocks. the attributes are standardized within Marquardt GmbH, although project-specific extensions are possible.

Usability of IRqA

Compared to simple text processing, the data base supported requirements management offers major advantages for Marquardt GmbH. As described in the chapters above, the various aspects (e.g. traceability, user rights etc. are well covered and performed by this tool.
The usage of a program like this takes some getting used to at first and sometimes appears a little cumbersome. The initial set-up and also large changes can utilize an import/export interface which offers the possibility to save time by exchanging data with standard Office solutions. In the Requirements Engineering Process the interfaces are to Microsoft Word und Excel as well as the convenient XRI (XML for Requirements Interchange). In the Test Scenario Engineering Process the selection of interfaces is limited to Microsoft Word und Excel.

The administration and viewing of various requirements in IRqA opens up a level of visibility which would never have been possible amongst the multitude of requirements in various documents, standards and norms. The displayed data records can be filtered according to attribute, text or such like. Via the previously mentioned block structure the data records can be displayed for just one or several blocks. By the setting of levels it is possible to show hierarchical views, assorted list views or reduced views dependent on attributes. Project-specific or user-specific views can be stored in the data base.

Test management

The technical content (test cases) of a requirement consists of a unique code, a descriptive name, the detailed test case description and its classification. This information uniquely describes a test requirement and using the link allows the coverage (traceability) to the customer requirements document and functional specification document to be shown. This is important for ensuring test coverage and for providing information, also for the OEM. Showing coverage isn't enough though for planning the scope, depth and time aspects of test execution. Apart from the test requirement's technical details the description therefore also includes information which better describe its planning and maturity aspects. This information is therefore also considered just as much a component of the test case's description as the technical inputs and needs to be stored with all the other information in the requirements tool.

For test process use Marquardt GmbH has divided the additional information into two categories. On one hand the planning information and on the other the life-cycle attributes.

Planning information
- Release
- Test phase
- Regression level
- Automated

These attributes are there purely to support the planning process and do not describe the functions or their content. In this way the release in which a test requirement will be checked is captured. Normally the number of test requirements grows in the course of project in proportion to the number of requirements. It isn't therefore possible to describe each and every test requirement right from the start of a re-

Wie überweisen Sie 5 000 000 000 000 US-$ sicher Tag für Tag?

Bei Geld verstehen wir keinen Spaß. Dafür verstehen wir, sicher damit umzugehen. So gewährleisten wir zum einen mit intelligenten IT-Lösungen die Sicherheit und Zuverlässigkeit beim Zahlungsverkehr US-amerikanischer Großbanken, zum anderen schützen wir fünf der zehn weltbesten Banken vor Geldwäsche-Geschäften.

Zwei von zahlreichen Gründen, warum wir zu den Top-Anbietern im internationalen IT-Service- und Consultingmarkt zählen. Und sollten Sie mehr Gründe erfahren wollen, was wir auch gut verstehen können, dann schreiben Sie uns einfach:

wow.de@logica.com

Releasing your potential
logica

lease. A further attribute is assigned so that the release after which the test requirement was introduced or the test execution in which the test requirement was performed can be documented. This procedure makes it possible to individually group the test suites (i.e. the test cases to be executed) for each test execution, which also permits retrospective traceability and, if where doubt exists, a repetition. The attribute can be an indicator of increasing test scope and also a marker for test cases which are no longer used due to changed requirements.
„test phase“ describes the type of test case according to the ISO/IEC 15504 standard. This standard requires various test cases for a new release. In order to avoid administering multiple test projects in the requirements management tool the test requirements in a project are allocated to a particular test. A test requirement can relate to a software test, a system integration test or a system test.

In the software test process time delays and unplanned bug-fixes occur quite frequently. This is taken into account by the test team when creating the test case by assigning a regression level for each test requirement. This additional information is not used during a regular test execution but can be used by the tester to prioritize the test should the entire time period planned for a test no longer be available.

When a test execution is being prepared for the next test it is important to know how many and which particular test requirements will be checked on which test harness so that these can be allocated to the various test systems. Do we need to check them on a fully automatic test system or do we need to perform some or all of them manually on a test bench? This information is of great significance for the detailed scheduling of the test execution because a manual test normally needs appreciably more time than an automatic test.
In addition this information provides a status regarding the reliability of the test results. The execution of an automatic test script is exactly reproducible compared with a manual test, which can suffer from variability of timing and sequence which under some circumstances can influence the results.

Life-cycle attributes

Up to now the elements of a test requirement described have related to functional information and planning attributes. For creating a test requirement, the development status also provides an indication of the test case's quality. On the basis of this maturity information it is possible to determine whether a test requirement is already available for a test execution or whether it still needs to be completed or changed. In order to accept a test requirement which has been created by the test designer for test execution it must first be submitted to a review. This entails explaining to the author of the functional specification and to software quality assurance why the test requirement was created that way and why particular attributes were allocated. Only then is it possible for the test case to be used for the test execution.

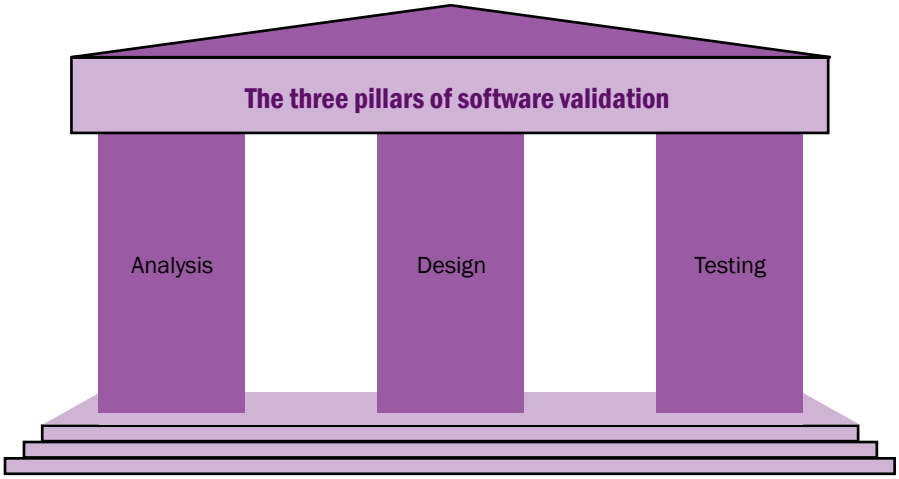
Values of development status are:

- In Work: The test requirement is currently under construction and its contents and additional information are not yet completely defined.
- In Review: The test requirement is ready for review. For release the author of the functional specification and software quality assurance must give their approval.
- Agreed: The test requirement has reached the highest maturity level and may now be integrated into the test execution.

Implementation status

This attribute is additional for test requirements which are to be executed in fully automatic test suites. The attribute captures additional information about the status of the automation. In a similar manner to the software development of control equipment, there are various states which must be transitioned and documented:

- is the test requirement already implemented in the test automation language?
- has the test input file been reviewed and released?
- has the test input file been put into operation?



The entered attributes make it easier for the person with test responsibility to plan the test execution and to properly justify the quality of the testing requirements to a third party. As a result of the data administration in the requirements management tool every project member can easily see the current progress, the test coverage, the status of the automation or the maturity of the test requirements. It is not necessary to check back on whether the documents in the version management tool are the right ones; we can rely on the fact that the current data can always be accessed in the requirements-management tool.

Test execution

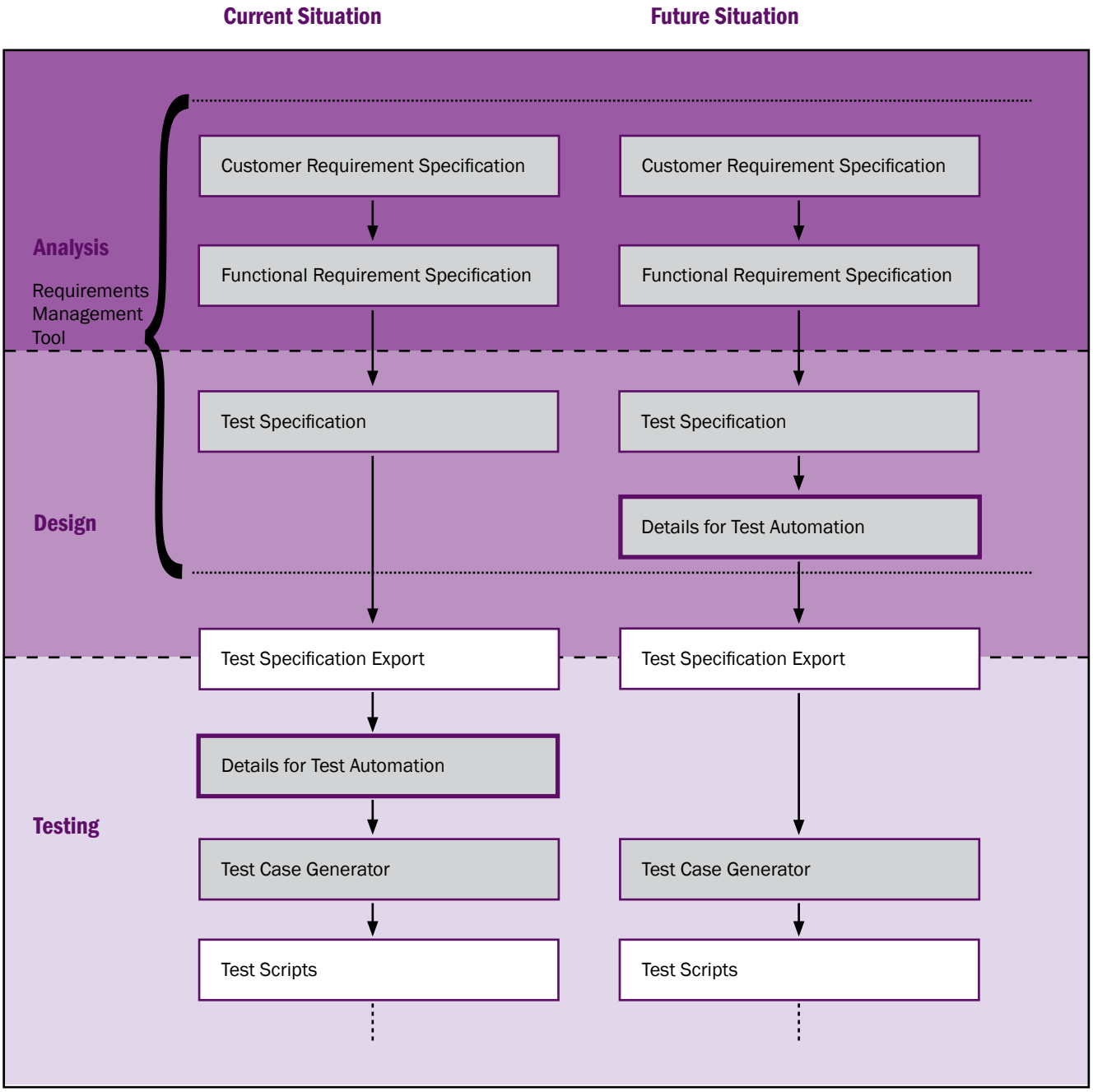
All information needed for the test execution and the creation of metrics can be exported from the requirements management tool as a protocol. This contains both technical and planning-specific attributes. The previously described filters or block filters are used to enable the export to be performed. During the test execution the protocol is supplemented by the automatic test system or the tester with information relevant to the test execution, in particular the results of the executed tests.
The information collected, test results and further attributes are then used for various metrics. These are based primarily on the entered test requirement attributes but also use the linking information to the requirements in the functional specification and, for example, the date of the last modification. All metrics are calculated and displayed without further effort so that they can immediately be made visible to the management, the project leader or the OEM.

Further development

For the software validation various process work packages have been defined which were developed according to ISO/IEC15504. These work packages will be continually extended and adapted in order to meet the future demands of the automobile makers and to be able to offer an even higher level of quality. To achieve this the interfaces between the three pillars of software validation will be adapted (see diagram) so that the Marquardt process always meets the current requirements.

For test case development a syntax has been defined so that a common test case title and description can be achieved. Using this standardized description it should be possible in the future to use a test case generator to create the test scripts for the various fully automatic test systems. In addition, the test specification will be exported from the requirements management tool into the test case generator and translated into a test script which suites the test system (see diagram).

The common syntax will additionally permit an easier introduction into other projects for every member of staff. All test case information is held in the data base supported requirements management tool.
In the future it is planned to use test case libraries which the test case designer can use when creating the test specification. Test cases which come into question for a library are



those which relate to test objects which can be standardised across several projects, such as those relating to the bus systems used, diagnostic functions, or the behaviour of the control units in various operational states.

An important and desirable aspect for testing would be the possibility to access the requirements management tool directly from the test application on the test bench in order that test suites can be executed. The data for the relevant test cases in the test run should be buffered in advance of the test or read on real time. After the test run the results would then be automatically stored in the requirements management tool. This would permit, amongst others, the automatic versioning of all test-relevant data in a tool.

These ideas should ensure that Marquardt GmbH can continue in the future to react quickly to the changing market and its increasing requirements in order to ensure the quality which the engineers themselves, the OEM and, last but not least, the automobile passengers deserve.

Authors: Norbert Ruck, Ralf Lulei, Thorsten Geiselhart, Gerd Rockweiler

The authors are responsible for the system test in the software validation at Marquardt GmbH.

Biography

Norbert Ruck studied Computer Engineering and graduated in 1996 as Dipl.-Inf. (FH). From 1996 until 2003, he worked as software developer in the development of control units for the automotive and medicine branches. Since 2003, he has been employed as software tester at Marquardt GmbH, where he is responsible, amonst other things, for the testing of safety-relevant control units. In addition to this, he is engaged in the creation, extension and training of the system testing processes and the strategic alignment of the test automisation.

Dipl.-Ing. (BA) Ralf Lulei is responsible at Marquardt GmbH for the system test for several large access and drive authorisation systems. Previous to this, he developed algorithms in the telecommunication area, with which automated test cases were developed from standardised protocoll specifications. Recently, he was part of the committee which designed the cross-project Marquardt test rig generation, with which automatic system tests will be performed in the future.

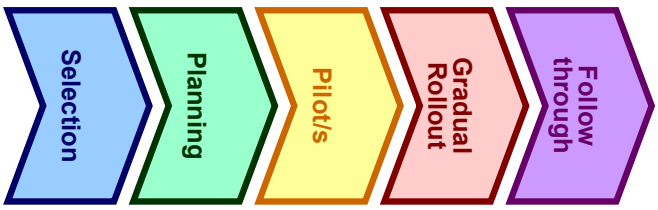
Thorsten Geiselhart concluded his studies at the Technical University of Ulm in 2004 as Dipl.-Ing. Electrical Engineering (FH). Since then, he has been employed in the software validation department at Marquardt GmbH. He is responsible for the system tests of several control panel projects, and, being a LIN expert, is contact person for his colleagues. In addition to this, he is responsible for the automation on a standardised test system. In this function, he looks after the test rigs and trains other Marquardt employees in the subject of automation, both in Germany and at the branch in Romania.
Gerd Rockweiler completed his studies as Dipl.-Ing. (FH) Computer Sciences in the degree subject communication technology. Since 2005, he has been working as systems engineer in the area of automotive software validation. At Marquardt GmbH, he designed an automated test rig for drive authorisation systems. As ISTQB Certified Tester Foundation Level, he is in charge of system testing and is system tester in various projects. In addition, he supports the test team with applications for test automation.

Implementing a new test management platform is not child's play, but it shouldn't be rocket science either...

by Joel Montvelisky

Test management platforms have evolved and became more complex over the past 10 years. What started as simple bug and test tracking tables have become intricate process and information management systems integrating the Testing Team with its Development and Project Management counterparts.

In the same way that the systems has become more complicated, so has the process to deploy and implement them. This may be the reason that today we see more and more companies that start deploying their platforms using only in-house knowledge and resources and end up calling an external consultant to finish the job for them.



Over the years I've seen my fair share of testing tool implementation successes and failures. If I had to sum all my experience up in one sentence it would be that: "*A platform implementation project should take into account the current and future needs of the teams and individuals that will interact with the system, regardless of whether they belong or don't belong to the team who owns the system itself*".

Planning & executing your deployment process

I will save you the painful evolutionary path it took me to refine my process and share with you the steps I follow (with good results so far) when deploying a new platform. As a guiding principle, use common sense and a bit of group psychology to guide your deployment and/or migration process.

I suggest the implementation of the following 5-step progressive process:

Step 1: Each organization is different, choose your platform carefully

Each test management platform comes with an intrinsic methodology, and the degree of customizations they allow is always limited. Take the time to investigate what tool is best for your organization. Download a trial version and test the way you can implement a process that suits your team around it.

Talk to other testers who use the tool to understand its strong and weak points better and faster. Look for them in places such as QAForums or ask your "sales representative" to put you in contact with someone in your geographical area; you will be surprised by the things you learn from talking to another tester instead reading a brochure or watching a demo.

Step 2: Plan your implementation and deployment all the way through

- Get mentoring from someone within your organization who has already done a similar project; her experience doesn't need to come from a testing tool, most Information Systems have many things in common you can leverage. If there is no one like that in your company you can again ask the "sales representative" (before you close the deal!) to contact you with someone in his organization to guide you through the process – look for help from either Customer Support or Professional Services, and if you are lucky (or a hard bargainer) you may get this service for free as part of your subscription fee.
- Get a Management Sponsor for the platform and its implementation process. It will help push the project through the annoying organizational red tape faster and provide more strength and legitimacy when convincing other teams to get into the migration process.
- Plan all the phases of your deployment up front. As the saying goes, don't go out on a trip unless you know how you will get back home safely. Plan for process risks that may occur, create avoidance and contingency plans for the eventualities that may arise during your deployment process.
- When planning the implementation and the methodology around the platform include stakeholders from all the teams who are related to the tool. Make sure these representatives are capable of making decisions for their teams. Include both the internal (testing) teams and all other teams that interface with the tool such as development, product management, customer support, etc.

We train in Spain

ISTQB Certified Tester Foundation Level

www.certified-tester.es

Díaz Hilterscheid

- Get support and participation from your IT team during the planning stage; make sure that you have everything technical covered in case something goes surprisingly wrong. Create a direct contact line between your IT and a technical representative from the vendor, don't waste your time been a dumb proxy that relays information and acts mainly as a process bottle-neck.

Step 3: Start the process with a pilot project

- Choose a pilot project correctly, looking for a team that is cooperative but also representative of your organization's needs. Sometimes you need 2 or 3 of these pilot teams, depending on the size and complexity of your organization.

- Hold periodic meetings to analyze the progress of your project. Use these meetings to review your issues and risks, and make the necessary fine-tuning operations. Aside from the periodic meetings make sure to constantly talk to users from all levels in order to get a good understanding of the satisfaction of your customers with the tool and process.

- Define exit and success criteria for the pilot. Follow up on these criteria constantly and communicate the results to your teams and stakeholders. Remember that one of the reasons for pilots is to gain organizational exposure and confidence in the process.

- Don't expect everything to go 100% smoothly; if things look too good to be true they probably are (too good to be true!). Encourage criticism and comments, learn from your pilot's results and implement the changes into the full deployment process.

- Use the pilot project(s) as your ambassador and sales team for all other teams in your organization. Take members of the team to help you sell the idea and benefits of the new system. Many times they will appear as non-biased on the project and this will help you to sound more reliable in the eyes of the rest of your organization.

Step 4: Production rollout

- Create a rollout list based on the schedules that suit the different teams. Take into consideration the needs and wants, as well as the process and cooperation levels of each team before migrating them into the new system. If possible, try spreading the teams over a period of time that will allow you to care for each team individually.

- Plan for training and mentoring of the users and teams. Training is the concentrated effort to introduce the team to the tool. Mentoring is the on-going task of ensuring the team is working effectively while answering their questions and handling their concerns, helping the users to feel comfortable with the platform. From my experience, mentoring is more important than training in most cases.

- Define point persons within each team to become your local administrators and representatives. They should handle the customizations, the hands-on training, and become the first line of support for their "local" users. Make these people your tool advocates and owners; this will save you a lot of effort in understanding the internal politics and needs of their teams.

Step 5: Follow-through

- Review the methodology as it evolves within your team(s). The use of a new tool will trigger people to think differently and improve processes that were stalled for years.Evaluate these changes and share them within your whole organization. I usually create a group composed of all the "local" administrators; this group meets regularly to talk about tool- and methodology-related topics and share issues and best practices.

- Join your local platform users' community. Again, look in QAForums and ask your vendor if there is a local community in your town; if there is no community, think about creating one!

- Keep in touch with your vendor and community. These guys keep improving their platforms and solving issues you may not even be aware you have. They are also a valuable place for sharing ideas and best practices you can implement.

Pitfalls to avoid:

At the beginning I mentioned companies starting their deployment projects using only internal resources and failing along the way. Surprisingly (or maybe not!) most of them commit similar mistakes that lead them eventually to either drop the whole project or call in an external consultant to finish the job for them.

Below is a list of the most common issues I tend to see at such customers. Even if none of the issues below by itself may appear to be fatal, their interaction and reverse-synergy cause the implementation project to slowly sink into a failure zone.

- **Installing a Production System on a Testing Environment.**

Most of us did it once or twice in the past, we took the "evaluation copy" someone installed in a testing server for the trial and continued working on it without moving it to a proper environment or even setting up a monitoring or back-up procedure.
The problems here usually start the day the server crashes, when you need an emergency rollback to restore data that was mistakenly erased, or when the first group of users starts complaining about the performance of the system and their inability to "work under the current conditions". Moving the server to a production environment at this point will involve complex data migration and most probably system down-time that will reflect negatively on the newly deployed platform.

- **Migrating your testing process "as is" into your new tool.**

All humans like routines (remember "Who moved my cheese?") but sometimes we need to ask ourselves if our existing methodology is both practical and efficient, or if there is a better way to perform our testing tasks and operations.
Even if you are convinced your current process is 100% effective and efficient, and regardless of what the sales representative told you while demonstrating the tool, every platform comes with its own defined methodology and no system is 100% customizable. During the implementation process you will need to make methodological compromises; not compromising and migrating your whole process "as is" will be like trying to fit a square peg into a round hole, at the end you may succeed but at what cost to both the square and the hole?

- **Letting each internal team define their separate process.**

Fact: *Unattended Democracy in the context of Information Systems quickly becomes Organizational Anarchy.*
Some companies like to smooth their implementation process by allowing internal teams the "freedom" to work on their separate environments and define their own working methodologies.
While small differences between teams is both allowed and even recommended, there will be a time when you will need to merge part of the testing or result data, to collaborate on tasks or to create a unified report for your top management (who'll want to take advantage of the expensive platform it let you acquire).
Unless you set a unified process where the teams work according to a pre-defined standard (and maybe even a project template) you will find yourself trying to merge oranges and apples or worse!
Try explaining the added value of an expensively unified platform to your management then…

- **Starting a Process Dictatorship as part of the tool implementation.**

The opposite of the Organizational Anarchy is also a known and common pitfall. The world is full of leaders who are sure they know what is best for their countries and their neighbors; we call them tyrants and dictators. The world is also full of *Tool Tyrants* who oppress their users and define the working methodology and standards without consulting with the rest of the team (btw, they are also convinced of doing this for the good of their organization).
When the testing tool becomes a tyranny, the users find ways to work around the tool. The

7th International Conference on
QA&Testing for Embedded Systems

International Conference on QA&TEST 2008 Embedded Software Testing

The Software Industry congregates in Bilbao

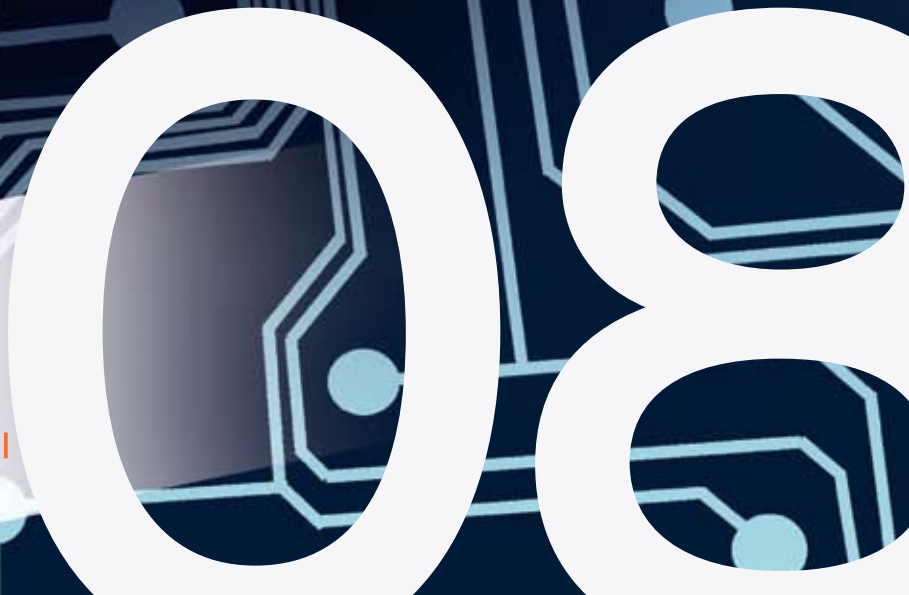
2 Tutorials
3 Outstanding Keynotes
20 Presentations divided into 6 thematic tracks
Interesting social events

Some topics of interest

- Human skills for testing
- Formal approaches to testing
- How to measure the impact of testing and QA in your oganisation
- Test and QA management

a.s.o.

For more information
please visit www.qatest.org



October 29 - 31
Palacio Euskalduna
Conference Centre and Concert Hall
Bilbao, Spain

best way for a manager to serve his users and expand the success of the tool is to treat them like paying customers, understand their needs and provide them with the solutions they need and like to use.

- **Biting off more than you can chew: deploying all your projects at once.**

Even if some managers like to do things all at once, there are only a small number of circumstances when all the projects in a company need to be migrated or deployed simultaneously; and this will always complicate the process. The best way to manage a deployment or migration process is by working with a representative pilot project followed by a gradual migration of the rest of the teams.
Each team requires proper help and attention when migrating their work and the lack of a good response generates stress and noise that resound on the platform for years to follow.
Unless it is absolutely necessary you will be safer with a staged and gradual migration.

- **Treating your deployment as a top-secret project.**

Some people think that if the deployment is risky and things may not go smoothly at first they should keep the program secret until they decide it is OK to tell. Here's the catch: nothing moves faster than gossip in a big company.
By keeping your project under a shadow of secrecy you will only create speculation and rumors that will damage what you are trying to achieve; allowing skeptics the opportunity to spread information that will make it harder for you to move forward with your mass deployment later in the game.

- **Becoming your platform's blind defender**

The platform you are testing, piloting and finally deploying is not perfect! It has bugs, it is not as customizable or dynamic as it could be (or as the previous system was), and it will certainly not have all the features everybody is looking for – no system ever does or ever will.
Always make sure to work based on the best interest of your company; your job is to select the tool, to design a way to deploy it, and to lead your organization through the deployment process.
Keep a critical and open mind (as far as humanly possible). When users complain about the tool and its functionality, listen to their issues, try to help them out and then pass this information along to the vendor; don't defend the tool as if it was your own - it will only make you look like you are getting money from the vendor's side.
Sometimes you will need to revise decisions taken in the past; avoid moving in one direction only because of the momentum of past mistakes.

Summary

Even as today's platforms become more complex and require more time and effort to perform a successful deployment you still don't need to be a tool expert to come out of your deployment alive. Just make sure that you follow a process that allows you to intelligently choose the tool and then gradually deploy it throughout your organization while making the necessary adjustments along the way.

Always concentrate on the best interests of your company, treat your users as paying customers who require your respect and care, and make your project visible and open to receive positive and constructive feedback from all stakeholders (internal or external). Last but not least, remember that you (and your teams) will need to make methodological compromises in order to match your process to the one supported by your platform – there are no 100% customizable systems.
When you work correctly your people skills become more valuable than your technical skills; and at the end of the process your organization will work more effectively and efficiently than before.

If you still feel you are running into invisible walls and/or your time and resource constraints push you to seek external help, don't feel bad or alone; depending on the tool (and the organization) it may still take knowledge and experience to successfully deploy a system. My advice then is to look for someone who will work with you (not instead of you!) helping to deploy the platform. Stay away from consultants who promise a fast, sterile and fully customized deployment; look for those who will help you analyze your needs and guide you and your team in your platform deployment process.

Biography

Joel Montvelisky has been managing testing and QA teams worldwide since 1998. As an ISTQB Foundation Certified practitioner, he has worked on multiple companies from Internet start-ups and all the way to managing the QA operation for one of the major Enterprise Test Management and Automation platforms in use today.

For the last couple of years, as an independent consultant, Joel has been helping companies to deploy and maximize quality-centered tools and methodologies, focusing on the processes to manage and improve the productivity and efficiency of the development and testing teams alike.

A relentless blogger around the subjects of Testing and Quality Assurance, you can read more from Joel at blog.quacktest.com

Díaz Hilterscheid

IT Management & Quality Services

www.diazhilterscheid.com

Traceable Knowledge by enhancing Requirements Management

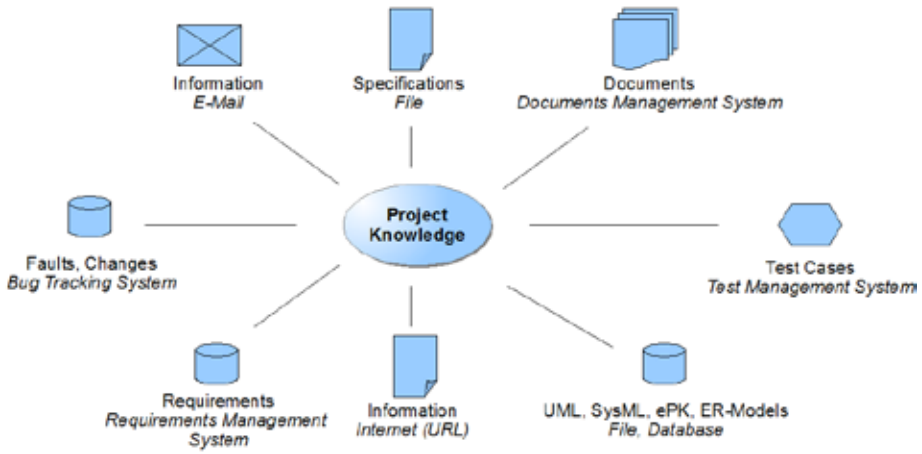
by Heiko Köppen

Abstract

Different studies come to the following conclusion: approximately half of all system problems can be explained by unsatisfactory dealing with requirements[1],[2],[3]. The appreciation of the concept of "Requirements Engineering" frequently confines itself to customer requirements. However, during the system development new or changed requirements, e.g. test requirements arise in the form of test cases or reusable library specifications as a result of value-adding processes. The storage of this information in different data repositories aggravates the transparency and the comparison of project knowledge as well as traceability, as demanded by various (safety critical) standards.(CMMI, aeronautics[4], common[5], railway applications[6]). Effectively dealing with this project knowledge (Knowledge Engineering) contains an enormous potential for improving quality in daily project work.

Every project starts with the customer requirements, which describe the system to be realized. But are the customer requirements complete and free of gaps, contradictions or redundancies? Open points are clarified with the customers during the project. The result is not always a new or changed specification or a change request. The informal way is often chosen. Emails, protocols or other information carriers complement the original customer requirements with new information. Regarding this for you as the responsible project manager the following questions arise:

- Are all project team members aware of the new information?
- What consequences will the new information have on the project (costs, time and quality)?
- Has the new information been taken into account in the project work, for example in the test?
- Are there supplementary protocols, documents, emails etc. which refer to the new information?



Picture 1: Knowledge comes from different sources

Integration of the information

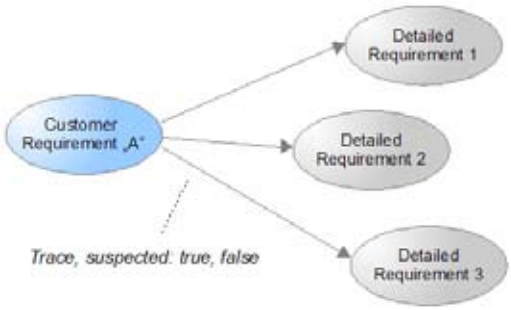
The reply to this question requires the coordination of the information from different sources by all project team members. The knowledge which lies in different repositories must be synchronized with a great deal of effort.

During a complex IT project a vast amount of information and knowledge arise which is relevant for the quality of the project results. It cannot be expected that the complexity of IT projects will decrease in future.

There is support at least for some part of the information, the customer requirements, with the requirements management. The use of requirements management tools makes it possible to organize, administer and maintain customer requirements.

It is a special advantage of these tools to represent the impact of requirements on other more detailed requirements. This means, for example, that a change to a customer requirement causes resulting traces to system requirements to be classified as suspicious automatically. Now, the system analyst now has the opportunity of adapting the system requirements according to the changed customer requirements. He is enabled to find out the effects of the changes on the project.

But a requirements management tool is just a spot solution for the system analyst. It is tailored to his needs exactly. The other stakeholders in your project will use different ones than this. This has the consequence that project information is distributed across many physically different data repositories. Retrieving



Picture 2: The trace - relation between source and derived requirements

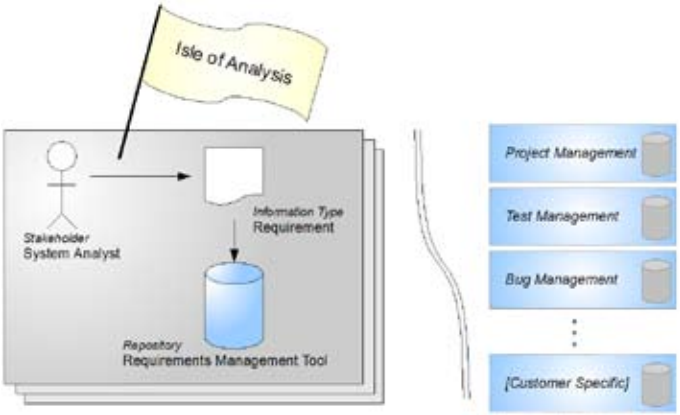
and deriving information is aggravated, the communication to different stakeholders is hindered. The representation of dependences between information from different data repositories and their maintenance requires a great deal of effort and therefore is not done in practice.

The Knowledge-Entity Concept

For this reason a new approach is necessary which supports the management of information in complex projects and provides success-relevant knowledge to every stakeholder. Through this the traceability and transparency of knowledge is guaranteed for all project team members.
The newly developed **Knowledge-Entity Concept** serves this purpose. Expressed in simplified terms: all information which has relevance for system development is structured, managed and subject to a life cycle uniformly. Knowledge entities are indivisible straps of knowledge or information. In projects knowledge is present in different forms e.g.:

- requirements
- test cases
- project planning tasks
- documents
- emails
- change requests
- links to external information systems
- models
- etc.

The contents of knowledge entities and their level of maturity are made transparent for all stakeholders. From this the realization degree of the project can be derived in detail.
Newly emerged information or changes to it become visible for the stakeholders concerned, consequences can be judged fast over all stages of development.



Picture 3: The spot solution for System Analysts

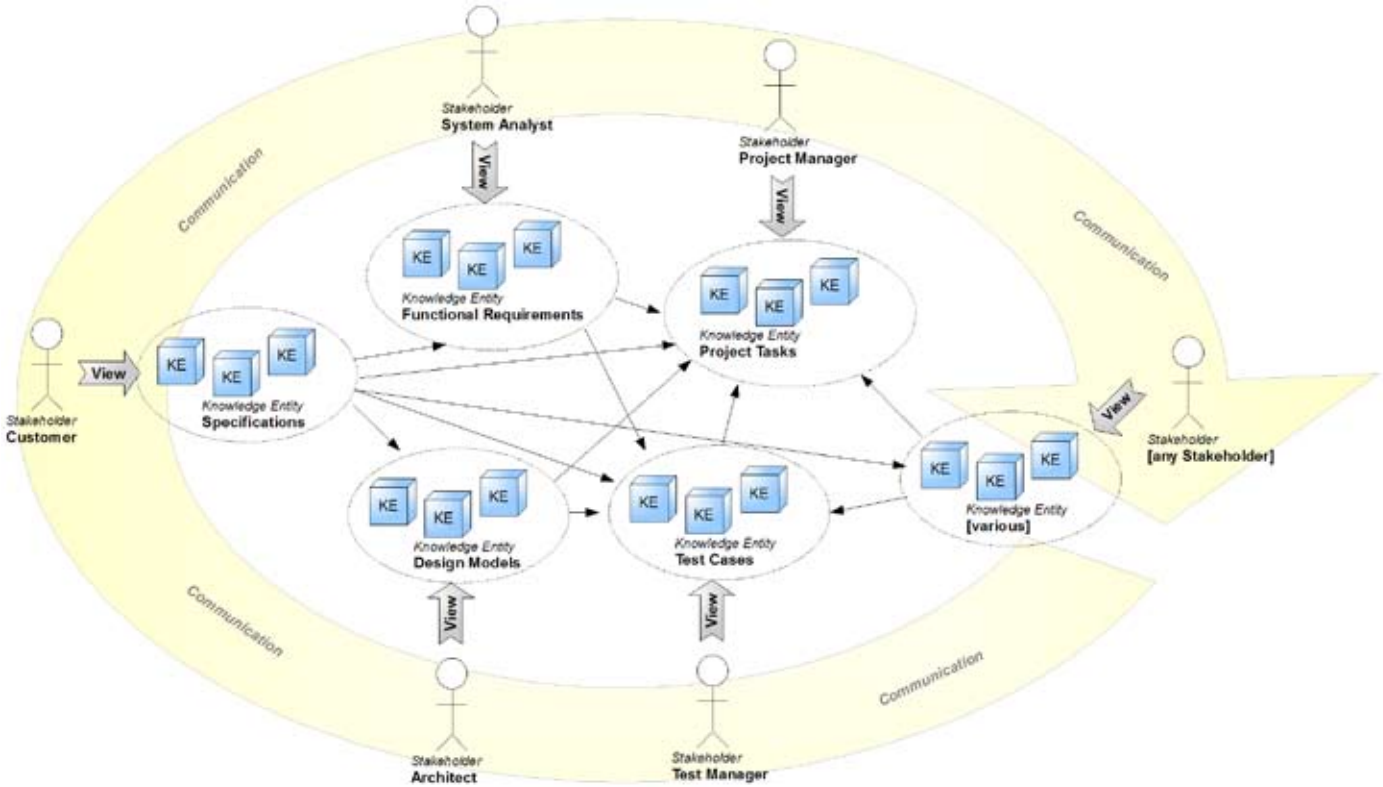
But it requires a tool to put the concept successfully into the practice. This tool must pursue a minimum of 3 aims, in order for the vision of project-wide traceability and transparency to become reality:

1. One information repository for all stakeholders
Instead of a tool-zoo, a unique tool has to be developed providing different, stakeholder-specific views on the information base. All information becomes

- visible (according to permission rules) ,
- comparable (using attributes) ,
- derivable (traceability), and
- controllable (workflow management,)

with this.

2. Integration of all stakeholders in the communication
Emails have the disadvantage that these are visible only for the sender and only for the receivers. For this reason an alternative communication method is required to support electronic discussions. Unlike emails, discussions should be made visible to all relevant stakeholders. In addition, they must have a reference to one or several knowledge entities.

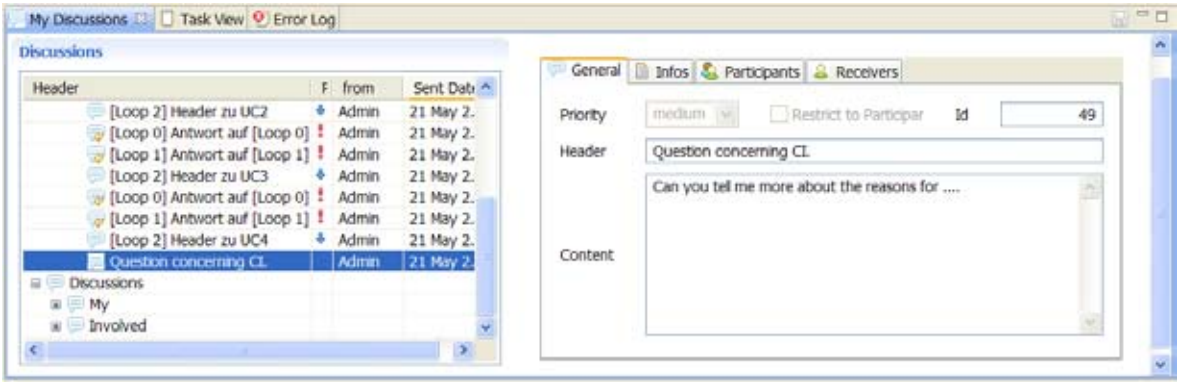


Picture 4: Traceability of Knowledge Entities & Stakeholder Communication

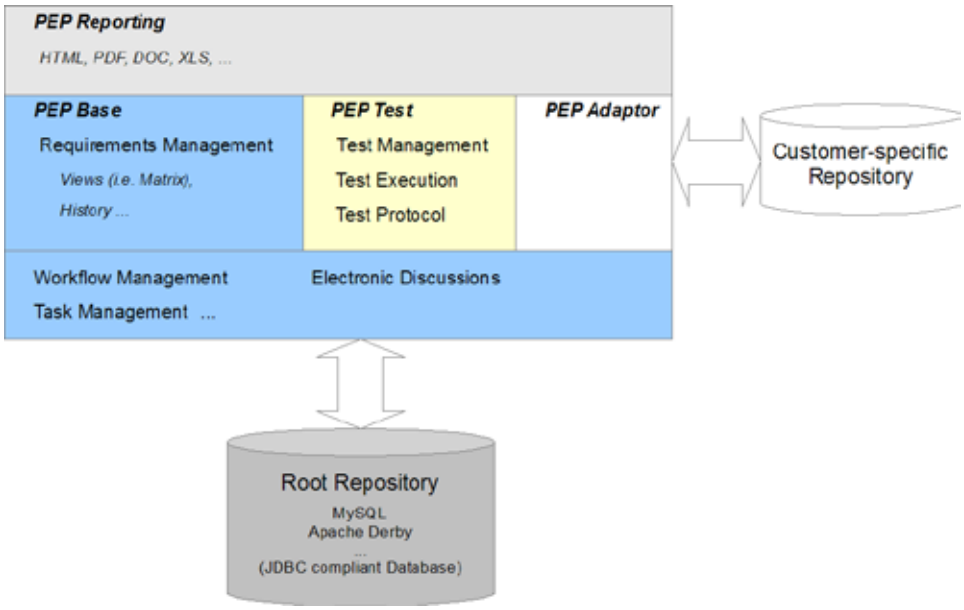
All persons involved in the discussion can receive emails about contributions to the discussion that have come in recently.

This is a possible way of how system-relevant information remains traceable and does not disappear in email nirvana.

3. Expansion of the term "requirement" on all stakeholders
Some process models already show that the idea of the request does not only confine itself to customer requirements. Requirements on the system can rather also be made by testers, for example (test cases). Every stakeholder can create project-relevant requirements from his/her point of view, which arises directly from customer requirements or arise from already derived requirements (e.g. architecture description, project planning task, test cases). It is, however, often difficult in practice to realize this concept because a suitable support by tools is missing.



Picture 5: The handling of Discussions in avenqo PEP is similar to emails



Picture 6: The structure of avenqo PEP

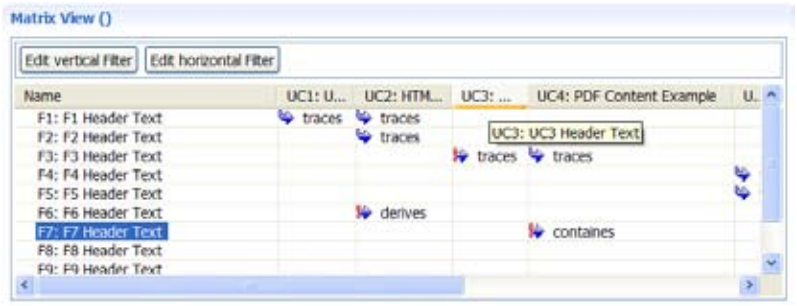
avenqo PEP

The support of the Knowledge-Entity Concept in practice as described above is the reason why the Project Engineering Platform (**avenqo PEP**) was developed. The system consists of a base module and a few additional plug-in modules with functional (stakeholder specific) extensions.

The system offers the advantages of a requirements management system and supplements it with stakeholder-specific functions and views. Currently, the views and functionalities of requirements managers, test managers and project managers are supported in the present construction stage. Managing faults or change requests is also possible using *avenqo PEP*. External tools can be added to the system using *PEP Adaptors*, a programming interface.

avenqo PEP Base
This module contains base functionalities for the administration of knowledge entities and to the support of the communication between the stakeholders involved. The module also provides functionalities which a user expects from a requirements management system: traceability, filters, matrix views and much more. In addition, *avenqo PEP Base* contains a workflow engine which allows the processing of knowledge entities according to user-defined processes (e.g. design, review, release).

avenqo PEP Reporting
With the reporting module various management reports can be generated e.g. about the realization progress, the test coverage or still open specifications. Furthermore, the export of knowledge entities (e.g. system requirements documentation) can be realized using different file formats (pdf, doc, html, xls, ...).



Picture 7: Matrix View in avenqo PEP shows suspicious Traces

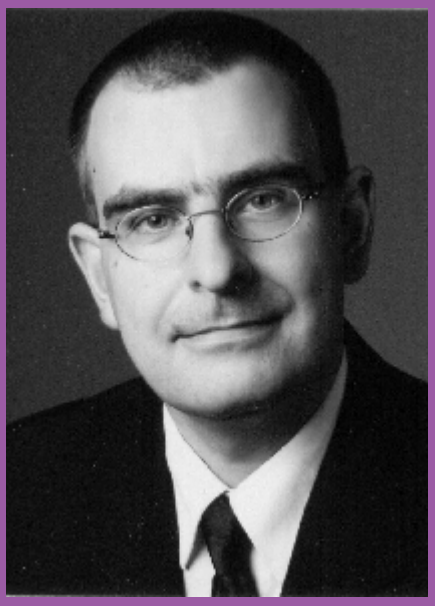
avenqo PEP Test
The module avenqo PEP Test was created for the special requirements of test management.
Based on the test cases created in avenqo PEP Base, dynamic test scenarios can be created and executed. The test results are stored in test protocols.

Further stakeholder-specific modules are planned in future. The software is built up modularly and therefore future-safe. You can download the free version of the software (Windows) from www.avenqo.com. Other platforms (Linux) can be supplied on request.

Summary

The integration of the knowledge of all stakeholders into the daily project work is the basis for the efficient and effective information exchange within projects. Uniting information from different stakeholder-specific data repositories and the management of its dependences is quite difficult in the practice. This was motivation enough to develop the avenqo Project Engineering Platform, which provides specific views and traceability on all project information for every stakeholder as demanded by various standards (e.g. CMMI). With avenqo PEP all project requirements and artefacts are organized centrally and can be subject to a workflow. With its email-based discussions avenqo PEP forms an excellent collaboration platform.

[1] Group, Standish: The Scope of Software Development Project Failures. CHAOS-Report. West Yarmouth, MA : The Standish Group, 2003
[2] Sheldon, F., et al. “Reliability Measurement: From Theory to Practice.” IEEE Software, (July 1992)
[3] Hall., T., et al. “Requirements Problems in Twelve Software Companies: An Empirical Analysis.” IEEE Proceedings-Software, 149, 5 (October 2002), pp. 153-60
[4] DO-178B, Software Considerations in Airborne Systems and Equipment Certification, published by RTCA, Incorporated.
[5] IEC 61508, Functional safety of electrical/electronic/programmable electronic safety-related systems
[6] EN50128 Railway applications. Communications, signalling and processing systems. Software for railway control and protection systems.



Biography

Heiko Köppen has been working in information technology for 16 years. Originally his main professional emphasis was on software quality assurance.
In many projects for SAP, VW and Siemens he has learned to appreciate the meaning of requirements engineering as a starting point for a successful project and test management.
His discontent with the requirements and test management solutions existing on the market led to the foundation of avenqo.

Treasuries from the dump - The software tester as an IT-archaeologist

by Dr. Konrad Schlude

Figure 1 - Wooden cups from the High Middle Ages, Stein am Rhein (Switzerland)[2]

Abstract

What has archaeology to do with software testing? From a simplified point of view, archaeology is about finding, securing, and interpreting human artefacts; software testing is about deriving test cases and running these test cases in a test environment. Whereas one of these topics is quite often "digging in the dirt" in not such nice weather and environment, the other is done in front of a computer in an air-conditioned office. The aim of this article is to show that the skills of a software tester can be used to do some kind of IT archaeology. Similar to an archaeologist who examines former dump sites and finds archaeological treasures there, we analyzed two areas of "dump data" of an IT system. With the results of this analysis, the underlying problems could be solved at the roots instead of fighting the symptoms. From the darkness of unnoticed garbage true treasures could be salvaged.

1. Introduction

It is possible to find treasures in the dump. Sometimes, archaeologists find very interesting things in former dump sites or mediaeval latrines. Especially in the deoxygenated climate of latrines, wooden objects can outlive several centuries. In their excavations, archaeologists have found mediaeval wooden cutlery and glasses frames. Outside this special climate, these things would not have survived the centuries.

So, a dump can give important information about life in former times [2].

1.1. Analysis of "Dump data"

Are there dark dump sites with hidden treasures in IT as well? Is there "dump data" that can be analyzed with profit? And what has testing to do with it?

In the following, we report on an IT project in which there was some kind of "dump data". Both in the development and production of an application data was accumulated and nobody took notice of the accumulation of that data. From our experience, this is the rule and not an exception.

2. The application and the development process

In order to better understand, we will give some details of the application and the development process. It is an application in the banking area. The customer company consists of several business units, and originally the application was developed for the prime business unit of the customer company. In the meantime, the application has become multi-tenancy, i.e., the other business units can work with adopted versions of the application.

The application can be used to choose a strategy from a list of proposed strategies, and the proposed strategies depend on the business unit (see Figure 2). Every night, batch processing checks whether the selected strategies are consistent with the current situation.

Besides an internal intranet version, there is an external version that can be accessed via the Internet.

It is a browser-based J2EE application with a mainframe component (CORBA, DB2, and batch) and a UNIX component (Oracle DB and batch). Therefore, there are several developer teams that are quite independent.

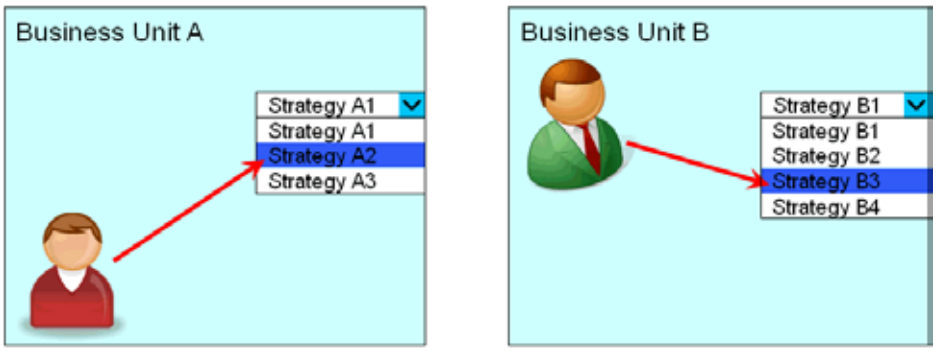


Figure 2 - Relationship between business units, strategies and clients

The application has been developed over a period of about five years, and there is further development. In the first phase, a lot of functionality was added. After that, the amount of new functionality was decreased, and therefore many team members left the project. Other external reasons led to "the great escape" from the project, especially in the requirements engineering team and in the JAVA team, many members left.

3. Error messages from batch processing

As mentioned above, batch processing checks the consistency of the selected strategies and the current situation. Shortly after this functionality had been deployed, the batch processing started to write error messages to a file. It was known that some of these error messages were related to external data delivery. Therefore, and since there was a lot of new functionality to develop, the error messages were ignored. Within two years, the number of daily error messages grew to 6,000. It became obvious that this number had to be reduced.

3.1. Results of the analysis

3.1.1. Changing the business unit

Most of the error messages (about 90%) were caused by changing the business unit. If a client is serviced by another business unit than before, this leads to an inconsistent situation (see Figure 3).

An automatic selection of a new strategy is not possible. In order to reduce the number of error messages, the application was enhanced to identify such cases on its own.

3.1.2. The -1 bug: quantum mechanics in IT?

A special problem was the so called -1 bug. In the database, there was one record with the strategy number -1 that stands for 'undefined', and this value -1 was not a legal value. Although there was only one such record, hidden in the large number of other problems, the problem was analyzed. This was due to the fact that the unknown reason could lead to other more serious problems. To analyze the situation, the developers were interviewed. The batch developers said that their program does not modify the value, the GUI developers said that there is a mechanism that blocks such illegal values. So, how could this value end up in the data base?

The situation set us thinking of quantum mechanics: a particle passes through a barrier although it has not sufficient energy for the passage (due to classical mechanics). Since the lead of the developers was a physicist, we called the -1 value a "quantum tunnelling'.

It took a long time to find out a way to reproduce the -1 value. It was a GUI bug, and in order to reproduce the -1, abstruse navigation through the application with more than 20 clicks was needed.

The bug was fixed by an improved consistency check. From a testing point of view, it was a very interesting bug as well. Quite often, developers respond to bug logs by claiming "But no user clicks this way!". The quantum tunnelling of the -1 bug is an illustrative example that users do everything they can do, although developers do not expect that.

3.1.3. Batch bug

For a special type of clients, the value of the business unit was updated in a wrong way and this led to similar error messages such as changing the business unit. In the beginning of the analysis, this problem was overseen in the large number of other error messages. This was one example of the well-known fact that a large number of uninteresting messages hide the interesting one.

3.1.4. Other bugs

Beside the problems mentioned above, several GUI bugs were detected just by clicking through the application. Especially the attempt to reproduce the -1 bug showed a way to generate inconsistent situations and exceptions.

4. Analysis of bug reports

Every year, there are several releases for maintenance and further development of the application. In each of these releases, several hundred bugs are reported and for defect tracking a tool is used. It is documented in which component the bug appeared, and for the JAVA part, the sub-component is logged as well. The number of bugs depends on the amount of new functionality. In one release with a lot of new functionality there were about 1,200 bug reports, in a maintenance release there were about 200 bug reports.

Before, there was no attempt to find a structure in the bug reports.

Important notice: Quite often, bugs are the results of cascades of misunderstandings. For instance, a stakeholder gives unclear explanations, the requirement engineer does not recognize this and writes unclear specifications, the developer does not recognize this and writes wrong code. In such situations, there is no "guilty guy" and therefore, it is the wrong attempt to evaluate the work of a developer by the analysis of bug reports. The fact that the analysis of bug reports should be done with regard to social relationship is a well known fact [1]: "It should be mentioned here that some questions can be very dangerous for the personal relationships with in the group and should, therefore, be addressed only in a very sensitive and objective fashion."

First, the bug reports were sorted by component. Most of the bugs were related to the JAVA component. Only a few bugs were related to the mainframe or UNIX, and there was no visible pattern. Therefore, the analysis was restricted to the bugs related to the JAVA component. Sorting by sub-component did not show any pattern, the bugs ranged from typing errors to serious exceptions. Note that this is an indication that the present division into sub-component is not helpful.

Another pattern became visible. Many bugs were neither regression bugs nor bugs of the new functionality; they showed up at the interface between a previous application and the new functionality. One example was the display of security numbers. In a previous release, it had been defined that the security numbers should be displayed due to the location of a user. However, the new functionality always showed the Swiss security number instead of ISIN or the German WKN.

This pattern could be found in other areas as well. For instance, the new functionality worked well for the internal version of the application, however there was an exception in the external version. For the various business units, similar effects appeared. The application worked well for the prime business unit; however, the implementation of the special requirements of the other business units was not correct.

27% of the bug reports were of this type.

Obviously, "the great escape from the project" had caused a loss of know-how. Both stakeholder and requirement engineers were focussed on the prime business unit, and therefore, the special requirements of the other business units were not specified well enough. The bugs were found in testing, since the test team was not affected by "the great escape ".

It is not surprising that the loss of know-how leads to bugs, however, the 27% showed that there had to be an improvement.

To reduce the number of such bugs, a checklist was provided. The most important "dimensions" were illustrated with examples. Especially, the special requirements of internal/external versions and the several business units are mentioned. The check list does not give answers; with the help of this check list, every requirements engineer, every developer, and every tester can ask the right questions to identify gaps.

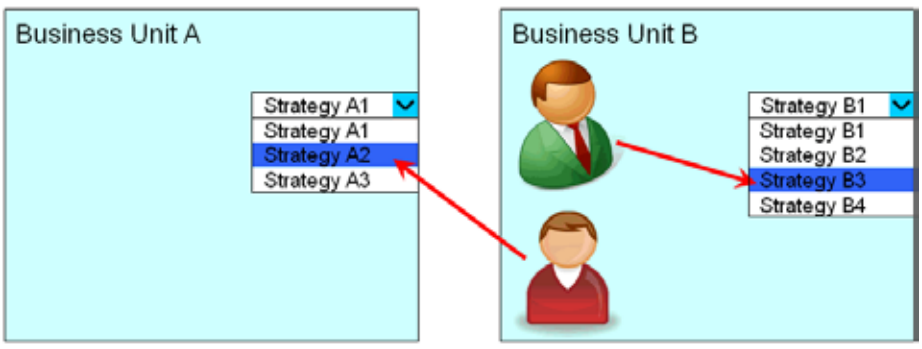


Figure 3 - Inconsistent situation after changing the business unit

5. Conclusions

Our conclusion is a contradiction to the notion of dump data: there is no such dump data. In principle, everything can be interesting and used to identify problems or to increase quality. In this article, we considered error messages from the batch processing and bug reports from the test process. In both cases, the sources of the problems could be identified and concrete suggestions for improvement were developed.

Most of the error messages were caused by external problems. However, several bugs were found as well. Whereas bugs in batch processing could be identified quite easily, the GUI bugs were hard to identify since there is large “distance” between the GUI and the error message of the batch processing. This is the reason why such an analysis cannot be done by a single developer. Typically, batch developers do not know the GUI, and GUI developers do not know the batch processing.

One reason for the importance of productive error messages is that bugs with low appearance priority appear in the mass test of the users. It is not surprising that the -1 bug was not found during testing, because no tester clicks that way.

Testing comes at the end of the development process. The result of testing, i.e., bug reports, can be used to gain information on the larger development process. This is similar to the error messages of the batch processing.

It is not a new idea that bug reports should be analyzed [1]; the well-known book [3] “How to break software” is the result of a bug analysis. Interestingly, analysis of bug reports is not part of testing education yet.

Bug reports and error messages are two possible ”heart rate monitors“ of an application. It is important to have such heart rate monitors. Another important aspect is to check these monitors.

5.1. Connection to software testing

There is one question to be answered: What is the connection to software testing? The answer is that for the analysis, typical tester skills are needed. One has to understand how the application and the development process work, the overview is needed. Furthermore, some “bug hunting passion” and the strong will to gain insight and to understand the problems are necessary.

From my point of view, the examples in this article illustrate how interesting software testing can be.

6. Bibliography

1. A. Endres:
   An Analysis of Errors and Their Causes in System Programs
   IEEE Trans. Software Engineering, Vol. SE-1, No. 2, June 1975, 140-149
   Kantonsarchäologie Schaffhausen (Hrsg.):
2. EX TERRA LUX: Geschichten aus dem Boden. Schaffhauser Archäologie des Mittelalters
   Schaffhausen, 2002
3. J. Whittaker:
   How to Break Software. A Practical Guide to Testing
   Addison Wesley, 2002

7. Wikipedia Links

- Batch processing: http://en.wikipedia.org/wiki/Batch_processing
- Multitenancy: http://en.wikipedia.org/wiki/Multitenancy
- Quantum tunnelling: http://en.wikipedia.org/wiki/Quantum_tunnelling

[2] Source: Kantonsarchäologie Schaffhausen, Switzerland [2]

Biography

Konrad Schlude holds the Diploma in Mathematics from the University of Freiburg and a PhD in Theoretical Computer Science from ETH Zürich.
Since 2004, he is with COMIT AG (Zürich) and works as a consultant in the area of software testing. He focuses on banking software.
Konrad is interested in test automation and has presented some results of his work at the 2nd “SAQ Software-Tester Forum” in Zürich in 2006.

Among his interests in mathematics and software testing, Konrad is an active member of the town council of Jestetten (Germany).

EDITOR
Díaz & Hilterscheid
Unternehmensberatung GmbH
Kurfürstendamm 179
10707 Berlin
Germany

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

Díaz & Hilterscheid is a member of "Verband der Zeitschriftenverleger Berlin-Brandenburg e.V."

EDITORIAL
José Díaz

LAYOUT & DESIGN
Katrin Schülke

WEBSITE
www.testingexperience.com

ARTICLES & AUTHORS
editorial@testingexperience.com

PRINT RUN
10.000 printed issues
approx. 25.000 downloads

ADVERTISEMENTS
sales@testingexperience.com

SUBSCRIBE
www.testingexperience.com/subscribe.php

PRICE
free of charge

ISSN 1866-5705

The Future of Software Security

by Prof. Dr. Sachar Paulus

Today's IT Security business is mainly dominated by the fact that there are companies that make a huge amount of money with the fact that standard software is inherently insecure. There are a couple of initiatives underway that will change this picture sustainably.

Increasing Regulatory Pressure

There is an increasing amount of regulatory action across the world to heighten the security of information technology. Some law makers are putting more focus on protecting information (such as e.g. California Civil Act SB 1386), others more on sound IT security management practices (such as the Japanese stock market requirements). However, the European Commission has started a public debate on responsibility for software security, and the main goal is to make the individual responsibilities of the different stakeholders (producers, service organizations, users) much more transparent than today. From there, market pressure will improve the security of standard software in a very short period of time.

One thing the industry needs for this is industrialization of security. That in turn means that we need to be able to measure improvements. Improvements of the security of software that we buy, improvements of the security of installed products, improvements of security management systems. But this is quite a new field.

Why Measure Security?

Why measure security? How does this work? What exists? This is still a wide open field. Most people in the IT business still understand "firewalls and virus scanners" if they hear about security. But security is more. Much, much more. Security encompasses the three attributes "integrity", "availability" and "confidentiality" of managed information. So basically, security means that "everything behaves as expected" - even under potentially malicious external influences. Now, for getting things better, we need to find a language that management understands. Today, they understand clearly demonstrated expenses for firewall licenses and administration. And they think the are safe. Far from true! With regard to the security of software, they fear evil, but are not able to make clear, justified decisions. So they either end up exposing critical information and processes barely protected to the Internet or they spend a huge amount of money in protection technology that does not solve the problem. We need to get to a state of measurable software security to enable business owners to make the right decisions.

Quality and Security

The relationship between quality and security is still not perceived in the right way, even among experts there is no one common, clear approach. Most people in the arena state that security "is just another quality aspect". But the reality is more subtle than this. In contrast to most other quality aspects which have been – either explicitly or implicitly – defined as expectations for the outcome of the product, hackers that take advantage of insecure software use unwanted features of the software. And this could then impact all three information security aspects: integrity, availability and confidentiality. So even a "Service Level Agreement" type of requirement for availability cannot be fulfilled if the software is stressed by external hackers. Therefore, when comparing quality and security, quality for me is "that the software does everything that it is expected to do", whereas security is "… but nothing else". The picture below demonstrates the difference. Of course, you can and must integrate security requirements into the quality assurance process, but it is important to understand that this often will not be sufficient for assuring software security completely.

Security in the software life cycle

So what is really contributing to secure software? It is a complex issue, and it affects all aspects through the software life cycle. SAP has started sharing its best practices with the community to raise the overall security of software installations. Here are the "10 principles" set forth by SAP. They have been presented jointly with Microsoft and Nokia to the EU Commission, which approved the approach as valuable.

1. During the requirements gathering process, the supplier shall define the security assurance properties of the product.
2. The design and the implementation of the product shall be reviewed whether it will not jeopardize the security assurance properties.
3. Before shipping, the product shall be tested whether the desired security assurance properties are present.
4. The software shall be able to run in "secure mode" after installation.
5. Security administration of the product shall be easy to understand and simple to use.
6. Necessary secure configuration of the software environment of the product shall be documented and actively communicated to customers.
7. During the product enhancement process, the supplier shall check that new requirements are not jeopardizing the existing security assurance properties of the product.
8. Software updates shall not jeopardize the security properties of the product.

9. The supplier shall implement a response process for effectively addressing security issues
10. Security issues detected in the product shall be communicated responsibly to users

Finally, there is one "meta-principle" which states as follows: *The supplier shall strive for developing measurement techniques for the principles mentioned above to continually improve the security of its products*. And this is exactly where we stand today and where the security community (who knows what to do to build secure software) needs assistance from the quality community (who know how to measure progress in software development) in order to find a language suitable for management to build solid decisions upon.

Measuring progress: why? and how?

Now, software is never 100% secure, as it will never achieve a 100% bug´-free status, except maybe in highly critical environments such as aerospace, aviation etc. where formal verification methods and product certification programs are used to prove the security – but note they are at the same time used for restricting the usage of the software to certain conditions, something we don't do with business software… So we need a way of constantly improving the effectiveness of the measures we employ. The key is measurability. This in turn will help us to prioritize the different measures against each other. And this applies both for software manufacturers as well as in-house development. Note for example that in principle 6 there is a hidden requirement to install a firewall if the application actually needs this level of protection (very, very likely today). So implicitly when measuring the success of applying the principles, the costs of these measures can be judged against other measures, allowing management to choose alternatives instead of blindly throwing money into the security expert's basket – or just getting paralyzed…

How to choose / evaluate a software supplier

One thing is to measure the success of measures to improve software security. Another is to actually evaluate the products and services of a software supplier. If you want to install and use a web server, it might be interesting to consider the security quality of the product as part of your choice. So you will actually need to compare the products – and the services of the supplier. Because it is not sufficient to look at the security testing results of a specific version, you also need to understand how well the supplier is prepared to react in case a new vulnerability is known. Even more importantly, the way how the supplier communicates and offers a solution (beyond patching, which is in practice often not possible right away) must be judged adequately. Again, measures and measuring techniques are missing.

Secure operations

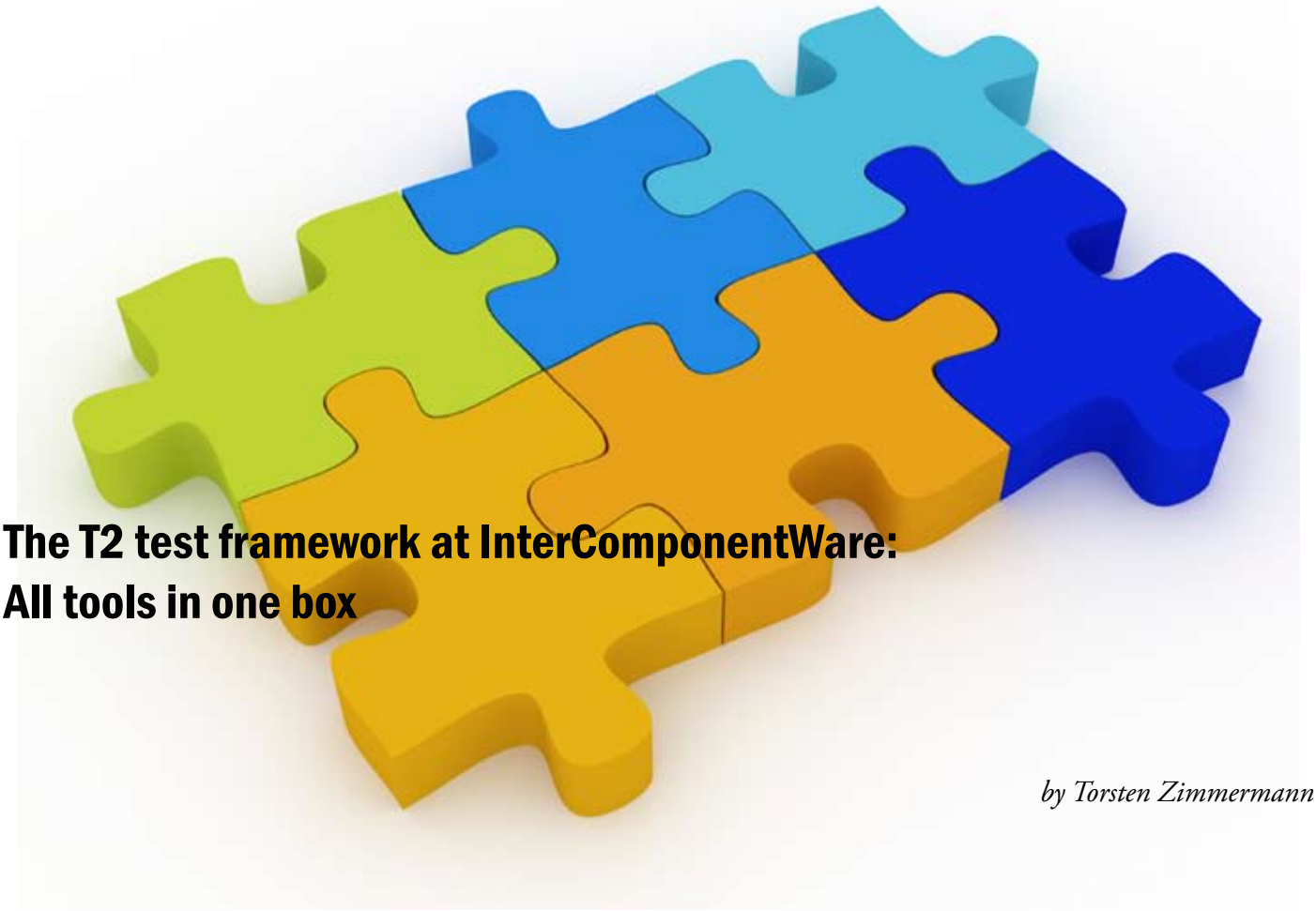
Finally, even if the supplier does a great job, we still don't know how secure the application actually is in operation. We need administration concepts, 24x7 support processes, change management etc., which all may affect the security of the process installed. There are certification programs and standards helping to build baseline protection, but this might not be suitable for addressing the specific needs of the business. Moreover, most organizations tackle these types of dangers from a risk management perspective, and thus quantitative analysis is needed to successfully integrate security management decisions into the business' risk management framework. Yet, there is little or no knowledge about measuring the security for a process, a business unit or even a whole company. There are security reporting and risk analysis frameworks based on generic risk catalogues, but especially in the software security arena this is still a very early field.

Summary

Measuring security, especially of software, is not yet on the agenda of most managers. But getting there will unleash a huge potential of savings, risk mitigation options by making management decisions possible. We, the security community, need help from the quality community for getting things done here. Only then will we ultimately be able to make software more secure.

Biography

Prof. Dr. Sachar Paulus is SVP Product Security at SAP and until recently he held the position of Chief Security Officer. Before this, he was Head of the Product Management Team for Security at SAP, coordinating security technology, secure development processes and security response for all SAP applications. Before joining SAP he worked as a consultant for SECUDE GmbH and KOBIL Systems GmbH, as well as doing research work at the Institute of Experimental Mathematics at Essen University and at the Department for Theoretical Computer Science at Darmstadt University of Technology. He has a PhD in Number Theory. He is recognized as a leading thinker in the security community and he is a member of several security organizations such as the Information Security Forum and the International Security Management Association. He is a member of the board of "TeleTrusT", a German non-profit organization for promoting secure electronic processes, a member of the Permanent Stakeholders Group of the European Union's "European Network and Information Security Agency" as well as a founding member of the "Application Security Industry Consortium". He currently co-organizes a masters degree curriculum on "Security Management" at the Technical University of Brandenburg, Germany.



The T2 test framework at InterComponentWare: All tools in one box

by Torsten Zimmermann

The quality assurance of applications is now more than ever a factor that is critical to success. In ensuring the quality of software it is important to follow a comprehensive and structured test concept. This must be an integral part of the design and development process right from the start in order to be able to exclude errors and performance problems from applications even before they go into operation – not an easy task for the responsible IT departments. After all, the development cycles for applications are becoming shorter and shorter, the legal framework conditions increasingly complex and the quality demands higher. As a result, the time and costs for the required testing are also under increasing pressure. Therefore the use of reliable, automated tools that can perform extensive tests quickly and inexpensively is indispensable for quality assurance.

This is precisely the challenge that InterComponentWare AG (ICW) is facing with its eHealth solution for the Health Service. The company specialises in four areas. These include telematic solutions, for example for the electronic health card, a solution for networking hospitals, solutions for doctors and doctors' networks, and the LifeSensor personal electronic health record. The quality demands on software in the Health Service are particularly high and are also governed by legislation, for example the Medical Product Act (Medizin-Produkte-Gesetz, MPG). In order to ensure the necessary quality of the applications, ICW has a central test platform at its disposal. This solution supports the complete process, from the definition of the tests to be performed to the commands within the tools used. This was previously impossible because there was no comprehensive solution. Instead, the work had to be carried out using a large number of tools in areas that were separated from one another. A comprehensive analysis on the basis of all of the test results was hence impossible.

Bus-system-integrated test tools

The T2 test framework (see illustration), circumvents the silo problem via a bus system that integrates the different testing solutions. The latter were selected by means of an elaborate procedure. To this end, ICW developed a requirement catalogue with an evaluation matrix, on the basis of which all of the tools were analysed by means of three different statistical methods. Among many other technologies, TestPartner by Compuware also supports the SAP area, which is very important for ICW, and offers advantages in object recognition and integrated languages. The TxBus enables the coordination and communication between TestPartner and the other tools and assumes control of the entire process.

The decisive requirement for this is that the test automation is not based on proprietary languages, but instead is independent of the tools licensed at ICW that have to be used locally. The test framework thereby forms the middle layer between the procedural test framework and the individual tools or the test platform. This procedure not only delivers comprehensive test results, but also reduces the time required for the tests. Using traditional capture/replay methods, around 380 test cases can be handled per tester in ICW's experience. Using the test procedure described, this number was already increased with the preceding T1 model to 4,800 test cases, which were performed within eight hours in an automatic run. Due to

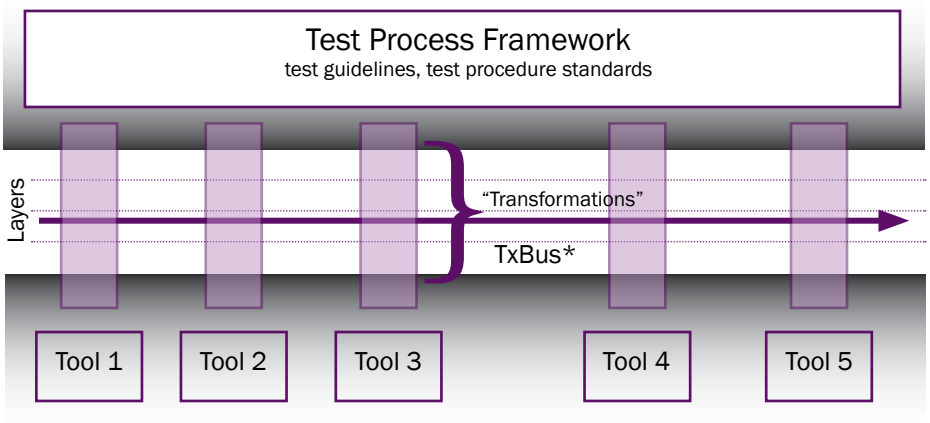


Figure 1

iSQI®
International Software Quality Institute

QAMP®

Quality Assurance Management Professional

Best prepared.

Certification to QAMP®

A certificate including broad scope:

Modular system of three certificates:

1st module:

Requirements Engineering (IREB® CPRE, FL)

2nd module:

Software Testing (ISTQB® CT, FL)

3rd module (domain specific knowledge):

Test Management (ISTQB® CT AL-TM)

alternatively: Configuration Management (iNTCCM®), Project Management (iSQI® CPPM), Software Architecture (iSAQB® CPSA), Secure Software Engineering (ISSECO®) (Other certificates are to be recognized by iSQI.)

A certificate including experience:

Two years of practical experience in software quality assurance management must be demonstrated

A certificate of value:

Annual re-certification ensures verification of further education and working experience

www.qamp.org

this increase, ICW now has a second method at its disposal, alongside risk–based testing, for closing the frequently existing gaps in test coverage and depth. ICW uses QADirector by Compuware here for controlling the test management. This is a web-based management console with integrated test solutions for planning, systemising and optimising the entire test process.

Clear dashboards facilitate the involvement of the business side

Besides the necessary technological flexibility, the solution must ensure right from the start that the software under development will also address the actual business requirements at the end. It is therefore necessary to involve the business side closely in the development process in order to define the requirements precisely. This can only be achieved if the tests are documented in such a way that they can also be understood and interpreted by decision-makers who have no technical background. For this purpose visual dashboards, which prepare the information comprehensibly and clearly, are planned to be integrated into the T2 test framework at ICW in future. In addition, if adjustments to the test process are necessary, for instance as part of an update, these can be represented and carried out more easily using dashboards.

Software development under the application of a comprehensive quality assurance strategy and the use of automated testing solutions is a complex process. This applies particularly since today new functionalities can be implemented very quickly using of frameworks. The expense pays for itself very quickly, however, since it is significantly cheaper to exclude problems and errors from an application before it goes into operation than to invest time and money in carrying out improvements on-site at the customer.

Biography

Torsten Zimmermann startet his professional career in 1993 after completing his degree as ‚Diplom Wirtschaftsinformatiker'. Since 1995 he has been contributing to international projects in the area of software quality and test management at various corporations, amongst them BMW, DaimlerChrysler, Hewlett-Packard and Hoffmann-La Roche.

He has developed the risk-based testing approach which has been amongst others presented in the professional publication 'QZ' and today is established as state-of-the-art knowledge in the area of software quality assurance. His accumulated experiences have led to the creation of the T1 TFT (Test Framework Technologies) which stands for the advent of a new area of testing systems.

Today Torsten Zimmermann is developing new approaches to enhanced test concepts and test frameworks. In cooperation with a network of universities this creates new solutions for rules- and model based testing systems.

Testing - the easy way

Before beginning to write a program – let alone test it – you've got to have a clear idea of what the program is going to be about, which needs it will fulfill and just how it's going to do so. That's where requirements engineering comes in.
Requirements engineering encompasses obtaining the right stakeholders to tell you what the program will be doing and where the constraints concerning time, budget and such lie. Furthermore, it involves helping these stakeholders to capture their knowledge – this may be done in natural language and/or using UML diagrams such as use cases, activity diagrams, class diagrams etc. This can be an arduous task at times, as the know-how these professionals posses has often become so "natural" to them that they find it hard to externalize. Here the requirements engineer has a plethora of techniques to assist him in aiding the stakeholders. Once this body of knowledge has been captured in a systematic fashion understandable to others – usually in documentation of some sort, it needs to be expertly managed, as it tends to grow and change till … well basically till the program is no longer used.

Now you might be thinking, that's nice and all – but what has it got to do with me? I'm just a tester and I don't really care if the program helps people farm carrots or deploy warheads, as long as it runs smoothly. Well, that might be true (although some people might have reservations about the warheads bit), but you've got to ask yourself: what do I test? Code. And what do I need to know about this code when testing it? Two things: 1. what it isn't supposed to do and – although often a bit neglected – 2. what it is supposed to do and within which limits is should do so. Well, that's exactly what well-written requirements will tell you. In fact, well-written requirements do more than that: they hand you the test cases on a silver platter.

Imagine a scenario where you don't have to agonize wondering if you've covered every possible or impossible way a future user might misuse the program. Where you don't have to rack your brain trying to come up with test cases which will ferret out every last little bug friendly programmers might have encoded for your personal delight. Where you know just how many tests you'll have to run beforehand. Sound good? Hot to get your hands on well-written requirements for your next project? Want to know how to turn them into test cases? Well, read on.

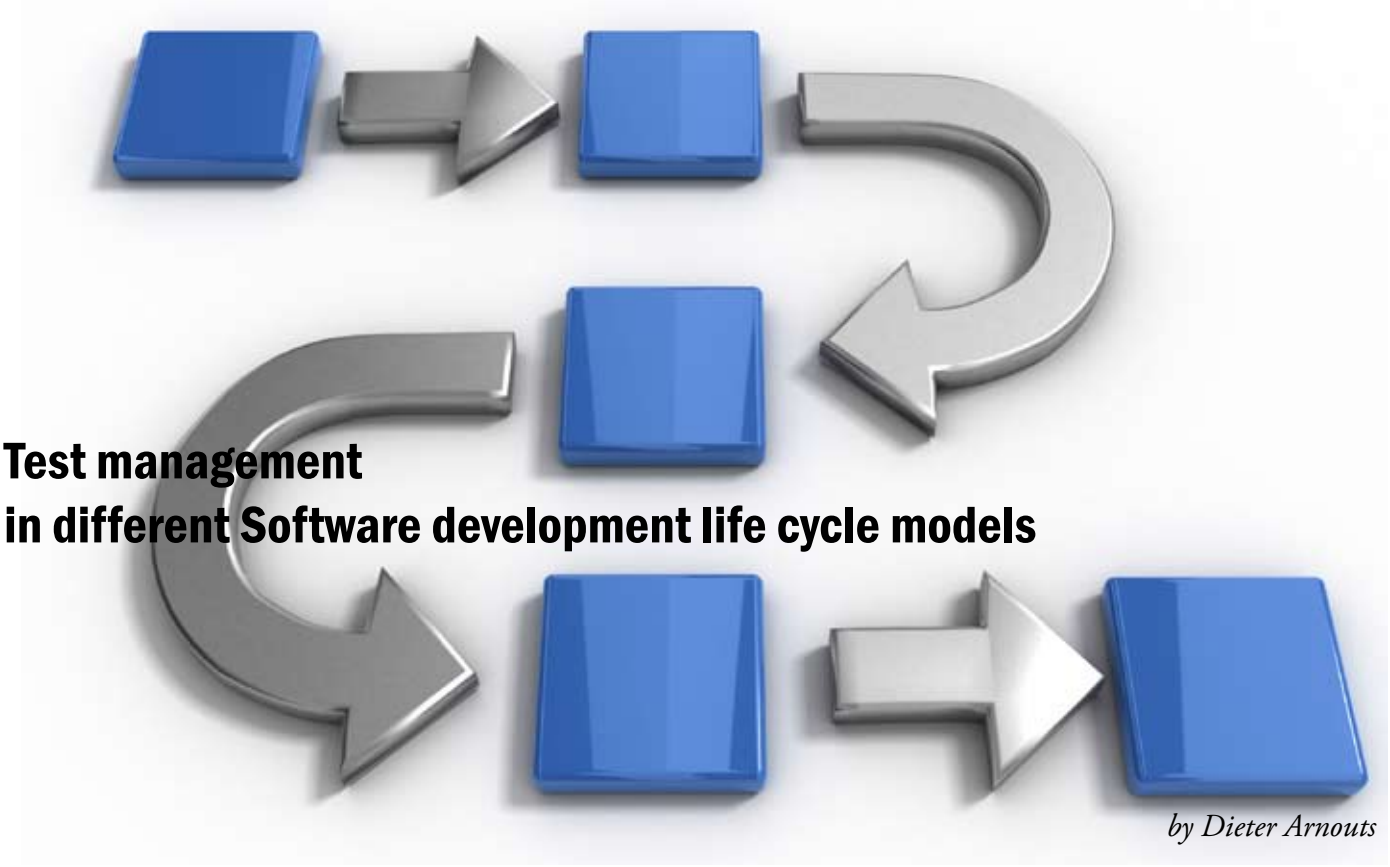
Because software development nowadays generally is a distributed venture where you might have management located in the U.S., stakeholders with the technical knowledge situated in Germany and the programmers doing their coding in India you need some way to guarantee that everybody will speak the same language when talking about the project. You need standards and you need to make sure that everybody's up to par. That's what the IREB (International Requirements Engineering Board) and the Certified Requirements Engineer certificate are all about.

The IREB (International Requirements Engineering Board) is an organization dedicated to collecting and managing best practices to do with requirements engineering and to standardizing and promoting information about the same. Amongst its board members are such renown professionals as Suzanne Robertson, Peter Hrushka, Klaus Pohl and its chairman Chris Rupp.

The IREB has created a certification called the Certified Requirements Engineer. This certificate is awarded after a three- or four-day seminar and a rigorous exam. To make sure everything is above board, the company offering the training and the certifier who provides and supervises the exam are always different companies - both licensed and overseen by the IREB. To guarantee that everybody's covered, seminars and tests are offered in Germany, Switzerland, Austria, the U.S., Israel, India, the U.K. and the Netherlands (with more countries up and coming very soon). In each of these countries at least one certifier is mandated to provide the test. Partners like the International Software Quality Institute (www.iSQI.org) are the nominated partner for some of the countries, but are also allowed to provide tests in countries not assigned to a certifying partner.
So if you're keen on simplifying the testing procedures in your next project, if you want to make sure you're handed documentation that is worth the paper it's printed on and makes your job easier rather than more complicated or if you simply want to know how requirements turn into test cases, you might consider having a look at:
www. certified-re.de or www. certified-re.com
There you'll find more information about the IREB and licensed certifiers near you.

Biography

Chris Rupp is the founder of NLP-based requirements engineering. She has developed and published pattern-oriented approaches to development. She is the author of numerous international publications and continues to work as a coach and consultant. Ms. Rupp is the founder and general manager of the SOPHIST GROUP. She is an expert in, among other things, natural linguistics and object-oriented methods, as well as organizational psychology and neuro-linguistic programming (NLP).



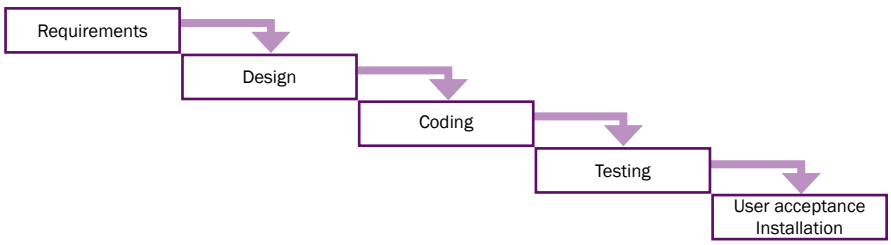
Test management in different Software development life cycle models

by Dieter Arnouts

Software development has gone through a lot of changes, different models have been and are being to build applications in the most efficient way depending on the requirements and specifications.
Each model has its own characteristics that influence the way of testing. Test management differs every time. Therefore an insight into the used development life cycle is important to guarantee qualitative testing output and therefore also a qualitative end product.
This article gives an overview of some of the most commonly used software development lifecycle models and how to adapt the test (management) process to development.

Waterfall

Software Model
The waterfall model is the oldest way to develop software. It is a sequential model, meaning that one product is developed and released to the customer with all agreed requirements in it. In this model another phase in the cycle commences after the previous one has finished. The largest drawback is that there is no flexibility available within this system. Each block is carved in stone before the process proceeds to the next phase.



Test Approach

Requirements	Development	Release 1	Release 2	Release 3	Release ...	Final Version	
	Writing testcases and testscripts	Full functional testing	Re-testing	Re-testing	Re-testing	User acceptance testing	Test Closure
		Integration testing	Regression testing	Regression testing	Regression testing		
		system (integration) testing	Defect handling	Defect handling	Defect handling		
		Defect logging					

Test planning in a waterfall model is quite easy. After the requirements are approved and before coding has ended, there is time to write test cases and scripts. Due to the fact that requirements in this model are completely fixed and detailed, test cases can be derived very easily and detailed scripts can be set up.

International
2008 SPICE DAYS
Software Process Improvement and Capability Determination

Topics

Telecommunication
Automotive/Aerospace
SPI Experience
Finance/Insurance
Assessments

Keynotes

Thomas Schlick, *VDA, Germany*
Richard Messnarz, *I.S.C.N., Austria*
Antonio Coletta, *DNV IT Global Services, Italy*

One-Day Automotive SPICE™ Upgrade-Training for Assessors

June 23 - 25, 2008
Andel's Hotel Prague
Czech Republic

25 % discount for Testing Experience readers

Register now!
www.spice-days.com

supported by:

in cooperation with:

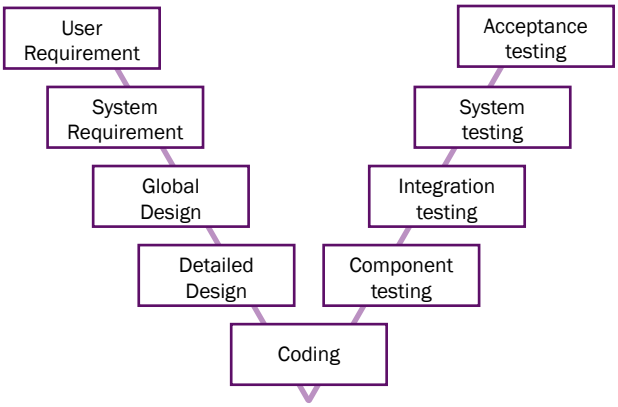
When test execution starts, a complete version of the software is available (Release 1), so testing builds up from simple functionality testing to system integration testing. Defects found are logged and sent to development. When testing of the first release has completed, development can release the second version. In this testing phase testing activities are limited since no new functionality has been added or changed. Only solved defects have to be retested. Regression testing can be important, depending on the severity of the solved bugs, but forms no major part of the activities. The same can be said for all the subsequent releases that should be available sooner given the decrease of open defects. Testing stops upon decision of the test manager or when all exit criteria have been achieved.

V-Model

Software Model

The V- model is also a sequential and can be considered as an extension to the waterfall model. The V-Model offers more flexibility to software development. The phases are not fixed and each phase, including the testing phases, can influence past steps. During this process requirements can change, which can lead to added or changed functionality.

Typical of the V-model is that each development phase has its corresponding phase on the testing side.



Test Approach

Requirements	Development	Release 1	Release 2	Release 3	Release ...	Final Version	
	Writing testcases and testscripts	Full functional testing	Functional testing rew specs	Functional testing rew specs	Re-testing	User acceptance testing	Test Closure
		Integration testing	Re-testing	Re-testing	Regression testing		
		system (integration) testing	Regression testing	Regression testing	Defect handling		
		Defect logging	Defect logging	Defect logging			
			Defect handling	Defect handling			

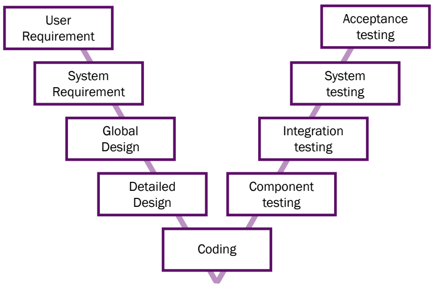
Early on, during the user requirements definition phase, test activities start by deriving use cases, which are useful for acceptance and system testing. The same goes for all next phases. Test execution however only starts after a full functional release is available. The problem and risks that go with the V-Model are that requirements can change since there is feedback possible between test phases and the development phases. New functionalities can be added or changed toward future releases. This can cause problems in the area of testing since test scripts have to be updated or rewritten in line with the new specifications. When functionalities are added or changed, regression testing becomes more important because there is a large likelihood that the changes have affected untouched parts of the software. Even so, testing with the V-model is feasible to manage, monitor and control. Please note that defect is a slightly more difficult than in the Waterfall model due to the fact that new defects can be logged in each new release.

Iterative model

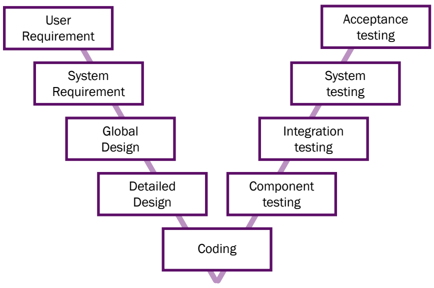
Software Model

The Iterative model is also known as a time boxed model. The software is delivered to the customer at fixed intervals. Each deliverable is a fully functional version as agreed upon. Subsequent iterations add new functionality to the system, so the system grows. Each iteration uses the V-model to build the software and fits into a timeframe. Another characteristic of this model is that each iteration has its own set of requirements that do not change, which is a big difference compared to the sequential v-model approach.

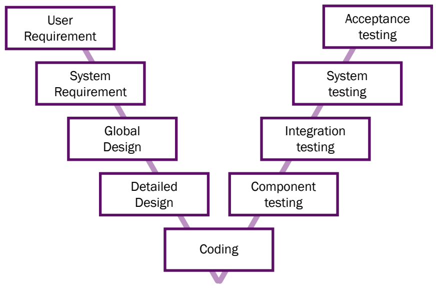
Final version 1



Final version 2



Final version 3



Test Approach

Requirements	Development	Release 1	Release 2	Release 3	Release ...	Final Version	
	Writing testcases and testscripts	Full functional testing	Functional testing rew specs	Functional testing rew specs	Re-testing	User acceptance testing	Test Closure
		Integration testing	Re-testing	Re-testing	Regression testing		
		system (integration) testing	Regression testing	Regression testing	Defect handling		
		Defect logging	Defect logging	Defect logging			
			Defect handling	Defect handling			

Requirements	Development	Release 1	Release 2	Release 3	Release ...	Final Version	
	Writing testcases and testscripts	Full functional testing	Functional testing rew specs	Functional testing rew specs	Re-testing	User acceptance testing	Test Closure
		Integration testing	Re-testing	Re-testing	Regression testing		
		system (integration) testing	Regression testing	Regression testing	Defect handling		
		Defect logging	Defect logging	Defect logging			
			Defect handling	Defect handling			

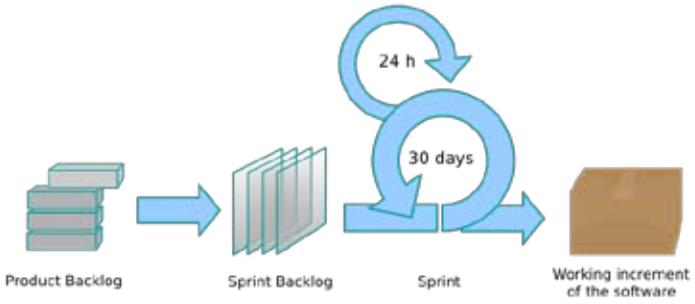
Requirements	Development	Release 1	Release 2	Release 3	Release ...	Final Version	
	Writing testcases and testscripts	Full functional testing	Functional testing rew specs	Functional testing rew specs	Re-testing	User acceptance testing	Test Closure
		Integration testing	Re-testing	Re-testing	Regression testing		
		system (integration) testing	Regression testing	Regression testing	Defect handling		
		Defect logging	Defect logging	Defect logging			
			Defect handling	Defect handling			

Testing for Final Version 1 can be considered the same as for the previously described V-model. From Final Version 2 onwards it becomes more complicated because you need to take care of open defects from the first version and start as early as possible with regression testing. In addition to this all the new added functionalities need to be tested, too. Maybe this does not take as much time as in the first version, but we have to take into account that regression testing demands more resources, so the test budget does not decrease.

Agile Model - Scrum

Software Model
The scrum model is an incremental model where sub-versions will be released after each phase instead of fully functional working versions. The Scrum process consists of sprints. Sprints are time periods of between 15 and 30 days. After each sprint a working increment of the software is delivered. This implies that when the sprint finishes, testing has to be finished, too. During a sprint, all activities run simultaneously so a specific approach for both development and testing is needed.

Sprint example



Test approach

Subversion 1	Subversion 2	Subversion 3	Subversion ...	Final Version (last subversion)
Requirements + Development	Requirements + Development	Requirements + Development	Requirements + Development	Requirements + Development
Functionality testing	Functionality testing	Functionality testing	Functionality testing	Functionality testing
Re-testing	Re-testing	Re-testing	Re-testing	Re-testing
Acceptance testing	Acceptance testing	Acceptance testing	Acceptance testing	Acceptance testing
Defect logging	Integration testing	Integration testing	Integration testing	Integration testing
Defect handling	Regression testing	Regression testing	Regression testing	Regression testing
	Defect logging	Defect logging	System testing	System testing
	Defect handling	Defect handling	Defect logging	Defect logging
			Defect handling	Defect handling

In this method all test activities have to take place in one sprint. There is no specific period to write test cases or scripts. Also, test levels are not maintained in the same order as in the V-model. To respond to this need, testers often use exploratory testing as a test technique. The danger by this technique is that documenting is missing or only partially done, but documented test cases are needed in later sub-versions to serve as regression tests. In agile methodologies, regression tests are by far the most important tests to execute given the nature of software development.

Conclusion

Test management is very dependent on the software development life cycle used. But nevertheless quality can always be guaranteed using the right planning, test techniques and appropriate defect management.

Biography

Dieter Arnouts (31 years) is Senior Test Analyst at Real. His specialties lie in adapting test strategies and management towards different software development life cycle models. He is also a dedicated trainer for software testing courses.

REAL is an IT business solutions and knowledge company with over 800 highly skilled IT professionals and more than 1,000 customers in the Benelux countries and France. The company offers business solutions and professional services designed to help its clients achieve their growth and profit objectives.

REAL specializes in providing innovative, cost-effective business solutions and IT knowledge in the following areas: Business Intelligence (BI), Customer Relationship Management (CRM), Web Solutions, Information Management, Enterprise Resource Planning (ERP), Enterprise Asset Management and Financial Accounting. For all of REAL's solutions it supports the entire software lifecycle: plan/design – build/deploy – run/maintain.

Dieter Arnouts started his career as an international trainer and technical support professional in the graphical industry, following which he made a career change towards software testing. In this area he has worked as specialist for several leading testing companies.

SEETEST
2008 SOUTH EAST EUROPEAN SOFTWARE TESTING CONFERENCE SOFIA

organized by:

QUALITY HOUSE

iSQI International Software Quality Institute

South East European Testing Board

July 2-3, 2008
SOFIA, BULGARIA
Kempinski Hotel Zografski Sofia

Topics:

Test Methodology
Test Management
Performance Testing
Test Automation
Test Process Improvement
Test Metrics

Keynotes:

Bj Rollison, Test Architect, Microsoft, USA
Ina Schieferdecker, Fraunhofer FOKUS, Germany

REGISTER NOW!

www.seetest.org info@seetest.org

supported by:

bait

SIGIST

ASQ

@nis

BASSCOM

media partners:

COMPUTERWORLD

sponsors:

hp invent

s&t

TRICENTIS

Testing Web Services with TTCN-3

by Theofanis Vassiliou-Gioles

Web Service technology is gaining more and more importance in the design and operation of new IT services. Therefore, a thorough testing of both single Web Service as well as whole services is a crucial factor for the complete business case.
Overall testing efforts can be substantially decreased if developers (with their related testing efforts) and quality assurance teams can rely on the same powerful test middleware capable of addressing their particular testing needs.
An efficient tooling enables developers to specify and execute their tests as early as possible, to run unit tests while coding as well as building a regression testing infrastructure.
On the other hand, testers responsible for testing complete systems rely on a technology that supports all components involved.

TTCN-3, the Testing and Test Control Notation specified and standardized by the European Telecommunications Standards Institute is an ideal candidate for providing such an accepted and deployed middleware test platform.

Some people may wonder how a test language coming from the telecommunications world could become suitable for testing software components and pure software systems. To answer this question it needs to be pointed out that TTCN-3 has been designed to provide a test technology that can be applied to different types of testing of communicating components. These can either be combined software/hardware components or pure software components, like interacting Web Service components. Although TTCN-3 has been designed primarily with black-box testing in mind, it is widely used not only in black-box testing but also in grey-box testing scenarios.

But how can TTCN-3 be applied to the testing of Web Services components? Web Service components in the present context are software components with interfaces defined using the Web Services Definition Language (WSDL) as standardized by the W3C ([WSDL]). One or more WSDL files define the exposed interfaces of the interacting software components, while so called 'bindings' define how these components communicate physically with each other. While not the only binding, the SOAP binding (Simple Object Access Protocol) is commonly used when enabling communication between Web Service components.

A Web Service-oriented software system (among other things) is built from one or more of these independent communicating software components. Figure 1 shows two software components that are communicating with each other via APIs (Application Programming Interfaces) while exposing different APIs to the outside world.

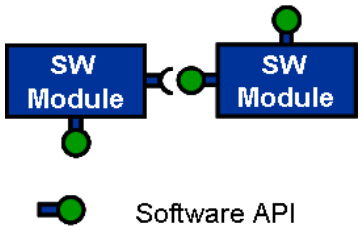


Figure 1 - Two Integrated Software Components

Even in such a simplified scenario different roles of software components can be identified.

- A software component acts as server, exposing an API and responding to operation calls.
- A software component uses the provided interfaces, i.e. makes calls to different software components, thus acting as client.
- With the combination of software components some interfaces are no longer accessible from the outside world, while the system behavior can be determined through other, exposed interfaces.

When testing such a software system, different questions have to be answered:

a) Does each software component provide the specified services at the exposed API? Even on invalid input?

b) How does a software component react if multiple components are attached to a single interface, or

c) How does a software component react if multiple components are attached to the different exposed interfaces?

From a testing perspective the greatest benefit can be achieved if one test technology can play all necessary roles (client and server) and is powerful enough to address the above-mentioned questions:

a) from functional, conformance testing with single components or modules,

b) over scalability tests of single components and modules,

c) to functional, scalability and load testing of a complete software system.

TTCN-3 has built-in features providing native tools in order to address above mentioned scenarios. The possibility of dynamic and concurrent test configuration using test components enables the testing of software components by simulating the behavior of one or more software components at the APIs. By using procedure-based communication for the APIs together with test data definitions implemented

as TTCN-3 templates, testers can concentrate on the essential aspects of testing, the 'what to test'. The way this is being technically implemented, while hidden from the tester, is well defined for the test system developer.

Figure 2 summarizes the different test scenarios from a test system perspective:

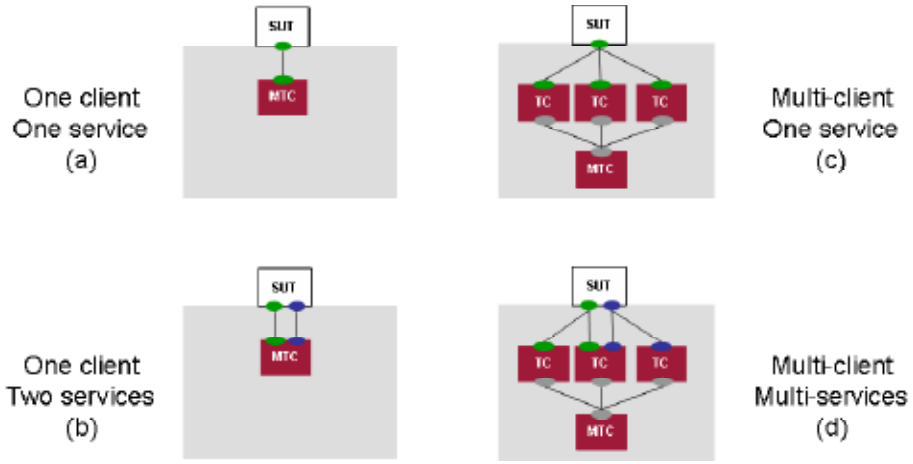


Figure 2 – Different Test Scenarios

From the functional testing perspective it is irrelevant whether the System Under Test (SUT) consists of one or more interacting software components. In the end, only the amount of and dependencies between the exposed interfaces are relevant to the provided services.

Configuration (a) displays an SUT which exposes only one interface to the outside world. In a functional test configuration the main test component (MTC) stimulates the SUT and assesses whether the SUT behaves as specified at this particular interface. In configuration (b) the MTC evaluates the correct behavior of the SUT in case two services are exposed to the outside world. In such a configuration the dependency between the interfaces is considered to be the important test aspect.

While the former two test configurations focus on the functional behavior of a software system at its interfaces, configuration (c) evaluates how the SUT behaves if multiple clients (here test components (TC)) access the same services of the SUT. In this case the MTC coordinates how the different TCs interact with the SUT, whether they perform the same or different usage scenarios on these particular interfaces. This type of test configuration is typical for a scalability or load testing scenario.

Finally, in configuration (d) multiple TCs interact with different services of the SUT, thus testing its full functionality. Again, the MTC acts as coordinator of the test behavior each TC component executes.

While so far we have discussed configuration aspects, let us focus on more specific Web Service oriented aspects now.

Figure 3 shows how artifacts from the system specification, in this particular case the interface specifications provided as WSDL description, ideally serve as the starting point for a test environment capable of testing the defined Web Services. In a TTCN-3 based environment, testers import the WSDL specification and are immediately able to start defining their test cases. The TTCN-3 scripts are based on the TTCN-3 binding of the WSDL specification. So on the one hand they make use of type and interface definitions derived from the WSDL specification, on the other hand they depend on the WSDL specification because the test system implementation has to respect this interface specification, too. Ideally, the test system implementation is totally transparent to the tester.

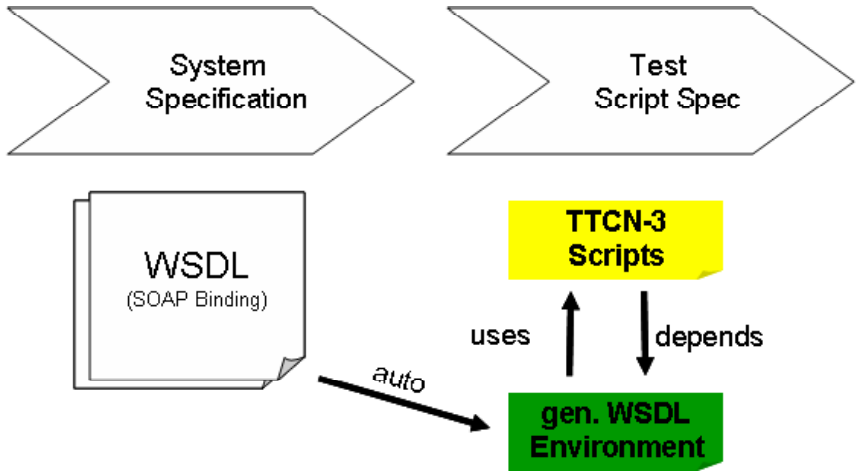


Figure 3 - From System Specification to Test Specification

WSDL to TTCN-3 Mapping

Different mappings for different technologies have been standardized with TTCN-3, particularly ASN.1 ([T.ASN.1]), IDL/CORBA ([T.IDL]) as well as a generic XML mapping (shortly to be published as Part 9 of the TTCN-3 standard). As no dedicated WSDL mapping has been standardized, Testing Technologies has filled this gap by defining a WSDL to TTCN-3 mapping which is made available as a generic TTworkbench plugin. Below, the mapping is explained with the help of a simple example, the Google Search API. Unfortunately the official support for this web service has been cancelled by Google. Nevertheless, its intuitive structure serves perfectly as an example for the introduction of the WSDL to TTCN-3 mapping.

Following code fragment shows a typical WSDL type definition. In the context of a Google search, the complex type **GoogleSearchResult** defines the response to a search query.

```
<xsd:complexType name="GoogleSearchResult">
    <xsd:all>
      <xsd:element name="documentFiltering"
type="xsd:boolean"/>
      <xsd:element name="searchComments" type="xsd:string"/>
      <xsd:element name="estimatedTotalResultsCount"
type="xsd:int"/>
      <xsd:element name="estimateIsExact" type="xsd:boolean"/>
      <xsd:element name="resultElements" type="typens:ResultEl
ementArray"/>
      <xsd:element name="searchQuery" type="xsd:string"/>
      <xsd:element name="startIndex" type="xsd:int"/>
      <xsd:element name="endIndex" type="xsd:int"/>
      <xsd:element name="searchTips" type="xsd:string"/>
      <xsd:element name="directoryCategories"
              type="typens:DirectoryCategoryArray"/>
      <xsd:element name="searchTime" type="xsd:double"/>
    </xsd:all>
</xsd:complexType>
```

By applying the WSDL to TTCN-3 mapping to this structure, an appropriate TTCN-3 structure - a TTCN-3 record - is created.

This simple example shows that the mapping from WSDL to TTCN-3 is very intuitive. Structured WSDL types are mapped to TTCN-3 structures, while the basic WSDL types are mapped to their basic TTCN-3 counterpart.

->

The interface description of the Web Service component for this particular service is defined by following fragment:

->

With a general understanding of interface specifications, it can be identified that the actual operation **doGoogleSearch** consists of the actual search query, with the parameters defined in the **doGoogleSearch message**. The response to the query **doGoogleSearchResponse** is described by the previously introduced and discussed **GoogleSearchResult**. After applying the WSDL to TTCN-3 mapping, the resulting TTCN-3 structures look as follows:

->

Looking into the mapping in detail, two aspects can be identified. First, testing of WSDL based Web Services uses the procedure based communication features of TTCN-3. The actual operation **doGoogleSearch** is mapped to a signature with appropriate parameters and return value. Furthermore, the WSDL specification defines that the Web Services can be accessed via a port called **GoogleSearchPort**. A TTCN-3 port **GoogleSearchPort** has been defined accordingly. The port is subsequently used to perform communication with the Web Service.

With this mapping, which is generated automatically in the appropriate tool environment, the framework for defining test cases has been created.

A typical test case for such a Web Service operation could have the following goals:

- Ensure that the Web Service is available and communication can take place.
- Ensure that for a given search key an appropriate set of hits will be delivered.
- Ensure that the response to a query does not take too much time.

Let us see how a test case checking these requirements would look in TTCN-3.

->

The test case **TestSearch()** consists of two steps: The first one sets up the configuration by mapping the **gp** port of the test component to the **gp** port of the System Under Test. The second step is the **doGoogleSearch** operation called with the parameters as defined in the template **gs**. Then the return value of the operation is matched against the data defined in the TTCN 3 template **expectedResults**.

```
type record GoogleSearchResult {
    boolean documentFiltering,
    charstring searchComments,
    integer estimatedTotalResultsCount,
    boolean estimateIsExact,
    record of ResultElement resultElements,
    charstring searchQuery,
    integer startIndex,
    integer endIndex,
    charstring searchTips,
    record of DirectoryCategory directoryCategories,
    float searchTime
  }
```

```
<message name="doGoogleSearch">
    <part name="key" type="xsd:string"/>
    <part name="q" type="xsd:string"/>
    <part name="start" type="xsd:int"/>
    <part name="maxResults" type="xsd:int"/>
    <part name="filter" type="xsd:boolean"/>
    <part name="restrict" type="xsd:string"/>
    <part name="safeSearch" type="xsd:boolean"/>
    <part name="lr" type="xsd:string"/>
    <part name="ie" type="xsd:string"/>
    <part name="oe" type="xsd:string"/>
  </message>

  <message name="doGoogleSearchResponse">
    <part name="return" type="typens:GoogleSearchResult"/>
  </message>

 <portType name="GoogleSearchPort">

….

    <operation name="doGoogleSearch">
      <input message="typens:doGoogleSearch"/>
      <output message="typens:doGoogleSearchResponse"/>
    </operation>
  </portType>
```

```
signature doGoogleSearch (
      in charstring key,
      in charstring q,
      in integer start_,
      in integer maxResults,
      in boolean filter,
      in charstring restrict,
      in boolean safeSearch,
      in charstring lr,
      in charstring ie,
      in charstring oe
) return GoogleSearchResult;

type port GoogleSearchPort procedure {
    inout doGetCachedPage; // REQUEST_RESPONSE,1
    inout doSpellingSuggestion; // REQUEST_RESPONSE,1
    inout doGoogleSearch; // REQUEST_RESPONSE,1
  }
```

```
// let's define on how to access the interface
type component SUT { port GoogleSearchPort gp; }
type component Client extends SUT { } ;

testcase TestSearch() runs on Client system SUT {
 map(self:gp, system:gp); // building the configuration
 gp.call(doGoogleSearch:gs, 2.0) { // call the Web Service
  [] gp.getreply(gs value expectedResults) { // wait for the
reply
    setverdict(pass) ; // pass if successful
 }
  [] gp.catch { setverdict(fail); } // fail if a timeout oc-
curred
 }
}
```

If the observed return value matches the expectation, the test verdict is set to *pass*. If the call to the Web Service does not terminate within two seconds, the test verdict is set to *fail*.

The test data for this test case look as follows:

```
template doGoogleSearch gs := {
  key := "TTCN-3 rocks",
  q := "",
  start_ := 1,
  maxResults := 100,
  filter := false,
  restrict := "",
  safeSearch := false,
  lr := "",
  ie := "",
  oe := ""
}
```

```
template GoogleSearchResult expectedResults := {
  documentFiltering := false,
  searchComments := ?,
  estimatedTotalResultsCount := (200 .. infinity),
  estimateIsExact := ?,
  resultElements := ?,
  searchQuery := ?,
  startIndex := ?,
  endIndex := ?,
  searchTips := ?,
  directoryCategories := ?,
  searchTime := (0.0 .. 0.2)
}
```

Conclusion

While this short example has only introduced the basic concepts of using the system specification artifact "WSDL" at test time, the full power of TTCN-3 can be applied with very little effort in order to achieve additional test goals, e.g. integration tests, scalability tests, etc. With a powerful test execution environment like TTworkbench, even more sophisticated test scenarios, for instance distributed ones, can be implemented.

As shown by this simple example, TTCN-3 based WSDL test cases can be easily designed and specified. A Web Service expert should be able to implement his/her own test scenarios based on the automatically generated test framework within a very short time. We have anticipated that especially non-software engineers appreciate this straightforward approach ([TOR08]). This enables the integration of Web Service components in test scenarios that go beyond the isolated testing of Web Services.

References

[T.ASN1]ETSI ES 201 873-7, Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 7: Using ASN.1 with TTCN 3

[T.IDL] ETSI ES 201 873-8, Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 8: The IDL to TTCN-3 Mapping

[TOR08] Torres, E. Pietsch, St.: Testing Web Services with TTCN-3; Software and Systems Quality Conference International 2008, Düsseldorf, Germany, April 2008

[WSDL] Web Services Description Language (WSDL) 1.1, 15.03.2001, http://www.w3.org/TR/wsdl

Biography

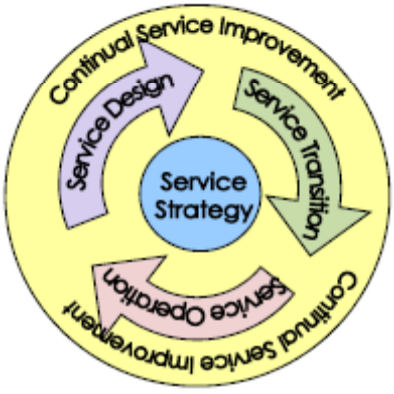
Theofanis Vassiliou-Gioles is co-founder and of Testing Technologies and has been its CEO since 2000. During his time with Fraunhofer Research Institute FOKUS for more than six years, he became senior expert in testing and was promoted to Deputy Chief of Competence Center for Testing Interoperability and Performance. Mr. Vassiliou-Gioles is involved in several standardization groups in ETSI, e.g. the TTCN-3 working group. He has published several papers on the application of TTCN-3 at various international conferences. Mr. Vassiliou-Gioles holds a Masters in Electrical Engineering from the Technical University of Berlin.

Test management with ITIL v3®

© iStockphoto

by Björn Lemke

ITIL® is a library with concepts for the management of IT services. Since the middle of the 1980s it has been developed by the British "Central Computer and Telecommunication Agency" (CCTA), today part of the "Office of Government Commerce" (OGC). The most recent version is version 3, which was published in May 2007. In version 3 ITIL® consists of five books and is built around the service life



ITIL v3® - Model

cycle. This is visualized in the model for ITIL® v3, built up like a spinning wheel. The axis is represented by the first book "Service Strategy". Around the axis there are the three spokes "Service Design", "Service Transition" and "Service Operation". All is framed by the tire "Continual Service Improvement" responsible for the grip to drive IT Service Management to the next level of customer satisfaction.
ITIL® v3 as a framework mainly uses the definition of roles, functions and processes and requires the use of Key Performance Indicators (KPI) to be measured and reported. The KPIs should be used for a transparent and fact-driven management of the processes. Even if ITIL® v3 covers the whole life cycle, it does not claim the need to be fully implemented, instead emphasizing that an IT organization should only use those processes that create value for it and its customers, even if there are many relationships between the processes. The biggest strength and at the very moment a weakness is, that ITIL® v3 only defines what to do, but not how it can be done. This requires specific tailoring of the framework to the particular needs of each IT organization, giving a lot of room for weak implementation.
But enough about the framework at large. Let´s have a look at the core publications and how the processes defined within them can be used for the management of the special IT service of software testing.

Service Strategy:

Service strategy supplies the strategic basis for the service organization. It aims at aligning the organization with the business needs. The scope ranges from the analysis of customer requirements through the definition of the service value recognized by the customer to unique selling propositions.
There are 10 questions central to the service strategy. These are:

- What services should we offer and to whom?
- How do we differentiate ourselves from competing alternatives?
- How do we truly create value for our customers?
- How do we capture value for our stakeholders?
- How can we make a case for strategic investments?
- How can financial management provide visibility and control over value creation?
- How should we define service quality?
- How do we choose between different paths for improving service quality?
- How do we efficiently allocate resources across a portfolio of services?
- How do we resolve conflicting demands for shared resources?

[1]

To answer the questions above, ITIL® offers a wide variety of supporting actions, like guidance on how to structure the organization or defining what criteria a service must fulfill to support the strategy. Central are three processes. These are:

- Demand Management: this process tries to understand customer requirements and brings them to action. In test management this process will help to determine what kind of tests and methods are required to fulfill the needs of the software customer.

- Service Portfolio Management: this process manages the whole service portfolio, including newly developed services as well as retired services. In this way it represents the whole set of capabilities offered by the service provider. It defines the unique selling propositions, strengths, weaknesses, opportunities and threats. Also the use of resources and capabilities. Therefore it could be used to define what test services could be offered to the customer. (Do we have the know-how and resources to offer load & performance testing, security testing, …?)

- Financial Management: this process quantifies the value of a service in monetary figures. It is responsible for the valuing, provisioning and accounting of a ser-

vice. In test management this process can be used for the estimation of ideal test coverage and how much the test will cost in relation to generated value.

Service Design:

Service design converts the service strategy into the "Service Design Package", which could be seen as "blueprint" of a service. It is responsible for a consistent, integrated and compatible design over all services. Compliance with regulations and standards is in the interest of the service design as well. The processes in service design are:

- Service Catalogue Management: SCM maintains the service catalog, the part of the service portfolio which is visible to customers (internal or external). A test organization can use this process to show its customer what kind of test and test management activities it can provide.

- Service Level Management: SLM defines and manages all service level agreements, operational level agreements and underpinning contracts. Service level agreements are the basis for providing services. They define and document the parameters of the partnership between service provider and service customer. For software testing this process will provide the customer with the necessary information about depth and scope of the tests to be executed, helping him to pre-determine if test end criteria can be met within given constraints by the service provided.

- Capacity Management: this process is concerned with planning and preparing the resources necessary to perform the service. Accurate long-term planning is essential, since resources not used generate a lot of cost without generating value, while a lack of resources limit volume and quality of services provided. In a test organization this process will define the needs of further test equipment, test analysts and other assets.

- Availability Management: having a resource doesn't help at all. The resource must be available at the planned time for the service to be provided. What value has a test laboratory if occupied by another project or down due to hardware outage? This scheduling of resources and other assets is the task of availability management. In a test organization this process can be used to determine which test manager, test analyst or laboratory will be assigned to the chosen service.

- Service Continuity Management: this process deals with the planning to provide the service, according to contract, in case of catastrophe by any means. This could include back-up plans for failed hardware (i.e. test server) as well as providing back-up data centers to prevent the loss of customer data or provide another test analyst in case of illness. The knowledge gained in this field could be reused as test oracle, predicting the needed behavior of the software under test in catastrophic scenarios. (i.e. the needed behavior if network connection is lost during a transaction)

- Information Security Management: this process is responsible for ensuring the security and confidentiality of data and thereby protecting the interest of all stakeholders. In a test organization this process can be used to determine whom to grant which kind of access rights to test systems, tools and test data for a given time to prevent accidental or intentional manipulation of the test setting.

- Supplier Management: supplier management is responsible for negotiating and ensuring the quality and performance of external assets. It ensures that the cost of the external assets is in relation to the value generated. Furthermore, it has to ensure the integration of the asset into its own process. An example for the use in test management is the evaluation of a bought-in test tool to ascertain whether it delivers correct results and performs as promised by the supplier.

Service Transition:

service transition plans and coordinates the going life of a new service. It integrates the service into the processes of the customer and ensures that no disruption of other services is caused by a new service. It is responsible for service usage as planned and for the fulfillment of customer needs. The processes in service transition are:

- Transition planning and support: this process is responsible for planning the resources and know-how necessary to perform a new service. If anything is missing, it has the duty to eliminate these gaps, i.e. by scheduling some training about the new used test methods.

- Service Asset and Configuration Management: SACM should provide an up to date description of all resources used, their relation with each other and their history. In the test management SACM can make sure the test is performed in a test environment according to the specification. Additionally, this process helps to run the tests always on the correct version of the software under test.

- Change Management: change management plans and manages all changes to a service and the environment the service is performed in. It integrates ever-changing business needs into the service life cycle by establishing new or changed services within production according to business needs and ensuring that no other service is disrupted. In test management this process can be used as valuable support for deviation management, allowing a controlled process to eliminate all deviations found during test.

- Release & Deployment Management: pre-packing, test and installation of a new or changed service is the assignment of release and deployment management. It is closely related to change management and also supports the deviation management.

- Service Validation & Testing: in this process the testing of services is done. It is an exception within the framework, since it provides a generic process flow. This process flow is close to ISTQB's generic test process. But since ITIL® is not about how things can be done it does not provide any methods to test software. Anyway the process still is useful for making sure a planned test service works as planned.

- Evaluation: this process has to measure if a service performs as planned. If there is any deviation between plan and performance the reason for this deviation needs to be understood and corrected. As part of test organization this process will help to see if tests progress as planned and to de-

rive actions necessary to return to plan if differences occur.
- Knowledge Management: the process of knowledge management is responsible for storing and sharing all gained knowledge. The knowledge should be presented in a structured way so everybody can find the relevant information. Therefore the "human factor" is essential in this process. For test management this process can be used to store all information and documents generated during test preparation, execution and post processing, like test protocols or lessons-learned documents. Structured storing allows easy access and reuse of all documents and thus helps to prevent a recurrence of errors. Test plan and test cases should be stored in the knowledge management system as well in order to be reused in regression testing.

Service Operation:

service operation manages the day-to-day operation of services and used assets. It is responsible for the effective and efficient delivery while at the same time allows for adjustment to changing business requirements. A key element of the service operation is the function "Service Desk", providing a single point of contact for all kind of user (and customer) needs and questions. The processes within this field are:

- Request Fulfillment: this process deals with all kinds of stakeholder request that are not incident-related. It does this by providing defined information channels between the service provider and the service customer. In test management this process can be used to provide the customer with background information about used methods or to inform him about current progress from regular reports if requested.

- Event management: event management monitors and manages all kinds of events. Events are occurrences within the infrastructure not causing a disruption of a service yet. Usually they are generated by automatic monitoring tools. For the software test this process is a supporting tool in two ways. First, it helps to make sure the test is performed in an environment according to specification and second, it helps to determine if recorded deviations are caused by the infrastructure or by the software under test. This is especially true for load and performance testing, where we have to make sure the bottleneck is not caused by the infrastructure in use.

- Incident Management: this process manages all incidents that occur while the service is being performed. The main task is to ensure that the service performs at the agreed level as soon as possible after the occurrence of the disruption. Therefore it ensures the best possible quality of service. During software tests this is the process to log all deviations and ensure the test progress is not delayed by the deviations. The logged incidents will ease the production of the test report as well as help to improve the used test processes.

- Problem Management: closely related to incident management, this process tries to solve the root cause of the logged incidents and other problems. Ideally it is proactive in solving problems prior to them causing disruptions. If we log all deviations during test execution within incident management this process will help to find the cause for the deviation. (i.e. error in the test concept or test data, error in software under test, …)

- Access management: also known as identity management. This process executes the policies decided upon in information security management and availability management. Therefore this process makes sure the test analyst/test manager has access to necessary tools and test objects.

Continual Service Improvement (CSI):

continual service improvement is the permanent effort to improve services, either quality-wise or cost-wise. All the other fields are embedded in CSI and therefore it gives a holistic approach to improvements over all stages of the service life cycle. For improvement, measurements are essential, since you always need to know your current performance if you want to improve it. However, it is not about improvement itself. The improvement always has to be in relation to the necessary investment (ROI for CSI [2]) and the advantages the business gains from them (The Business Question for CSI [3]). Besides proposing a lot of process improvement methods known in general management literature ITIL® v3 suggests its own methods. Examples are:

- the CSI Model: the CSI Model is a loop containing six questions. The loop is permanently repeated. The questions within the loop are:

  - What is the vision?
  - Where are we now?
  - Where do we want to be?
  - How do we get there?
  - Did we get there?
  - How do we keep the momentum going?

[4]

- The seven step improvement process: This process is organized in a loop as well. The center is formed by the identification of vision, strategy and goals aimed for, with the process surrounded by the following steps:

  1. Define what you should measure
  2. Define what you can measure
  3. Gather the data
  4. Process the data
  5. Analyze the data
  6. Present and use the information
  7. Implement corrective actions

[5]

Conclusion:

As you have seen, all of ITIL® v3 could be used for the special IT service of software testing. So regardless of whether your test organization is part of a big IT shop already using the ITIL® proposals and you want to integrate better into it, or a small test consultancy looking for a framework to improve your performance, give ITIL® a chance to be your inspiration. Pick the processes delivering quick wins or eliminating weaknesses and see where it can take you.

Bibliography:
[1] ITIL® v3 – Service Strategy; Page 27; OGC; 2007
[2] ITIL® v3 – Continual Service Improvement; Page 132; OGC; 2007
[3] ITIL® v3 – Continual Service Improvement; Page 141; OGC; 2007
[4] ITIL® v3 – Continual Service Improvement; Page 127; OGC; 2007
[5] ITIL® v3 – Continual Service Improvement; Page 68; OGC; 2007

Biography

Björn Lemke works as consultant for Díaz & Hilterscheid Unternehmensberatung GmbH. In this function he is mainly involved in ITIL® related projects.
He started his career as scientific assistant at the University of Applied Sciences Furtwangen, Germany. Simultaneously he worked as freelancing project engineer. During this freelancing he met his former employer, a mid-size IT hardware manufacturer and distributor. There he was employed as System Manager for more than 5 years.
Björn holds engineering degrees of the University of Applied Science Furtwangen, Germany and the Hanzehoogeschool Groningen, Netherlands. Besides he is ISTQB Certified Tester Advanced Level – Test Manager and holds the ITIL® v3 Foundation Certificate.

Mind the gap between Test Strategy and Execution

by Erik Runhaar

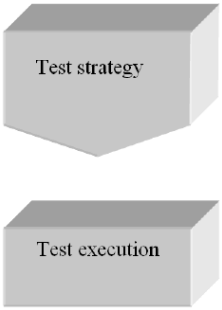
© iStockphoto

"We want 100% coverage of all functions. This means 100% service readiness, overall". This was a statement I heard an airplane-factory manager say to the R&D department of an ERP software house. "This is what we do within our airplanes and this is what we expect from you as well". It looks like a reasonable acceptance criterion and, especially within an airplane-factory, we believe people accept this responsibility. It is a logical statement at an employee meeting where pep talk is the main goal. But if we analyze what this woman really tried to make clear, we cannot distil anything useful for testing out of this statement.

This is exactly the problem we are facing when dealing with the issue: what do you mean when you are talking about coverage? And what is the link between this coverage and test execution and strategy?

First of all I will illustrate the problem with the following example, which might be recognizable.

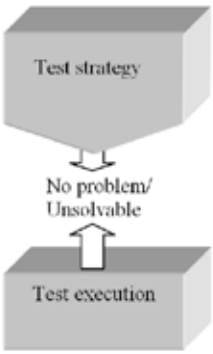
The project has started and Jan is invited to the Project Start Up meeting. In the organization Jan is working for, testing has the right place early in the project. Directly at the start of the project, Testing is involved and represented by the overall test manager (Jan). After the meeting Jan starts to collect product (and project) risks and priorities in a stakeholder meeting. He invites development as well as support, the customer, testing and other groups on the impact side and the likelihood of failure side. The meeting results in a test strategy in which the most important risks that were raised in the meeting are covered. Jan finishes his Master Test Plan. The document is approved by the customer. So far this project runs in a way many test managers could only dream of.
Then the test analysts are involved. They start reading the functional design and create their test scripts (test procedures). They do their utmost to cover as much as possible of the Functional Design. They even use the two test techniques (equivalence partitioning and error guessing) they learned during a course. In some cases they switch to the method of the tracer, marking all important parts in the design. If you were able to talk to them, and ask how they know what to test (in other words: how they implemented the strategy), their answer would probably be: "Oh, if time wasn't that scarce…"



This example illustrates the challenge many companies sooner or later have to face. There is a gap between the strategy and what the testers do in script preparation and execution. In my experience, the main reason for this is that the testers in most cases are not involved in the strategy meeting. They are not aware of a strategy at all defined by the stakeholders in the company, including the customer. In other cases, there is a strategy and the test manager and the testers are aware of this document, but none of them link this strategy to the way of testing, or are aware that that strategy should determine test execution. The following question arises: what do you report in a situation, where the test strategy reflects the customer's requirements for quality, but the test scripts just aim to cover as much as possible, because you never know…

Issue or not

Today IT and Testing is becoming more and more "visible". On the one hand customers want the software to be 'defect-free' (a persistent misunderstanding). On the other hand, the customer wants the same software preferably yesterday and for a limited budget. That is the problem IT faces nowadays. As a reaction to this, agile testing methods and exploratory techniques arise as the solution: shorter time frames, shorter lines between customer and IT and Testing, no bureaucracy. No worries about issues like test strategy and test execution or even a relation between the one and the other.

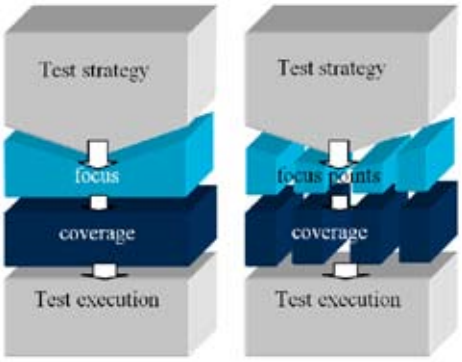


I notice in my daily work as a professional tester, that many companies do not even recognize this as an issue. In practice, at some point, it will hit the organization in the face. Traceability of the result and risks covered into test execution is not possible.

Another category of person is convinced this problem is unsolvable: "We have to live with it, we have to work in short time spans and with low budgets. So what choice do we have?" Especially in the latter situation, the answer is that the indication of scarce time and money turns out to be the key to solve the problem.

The clue

Time and money, these are the levers to link the customer's requirements and the results of test execution.
A concept is needed to get the best testing done (cost) in the available time (time). This concept deals with what to test (focus). This focus must be determined by the customer of the product. This customer has a long list of

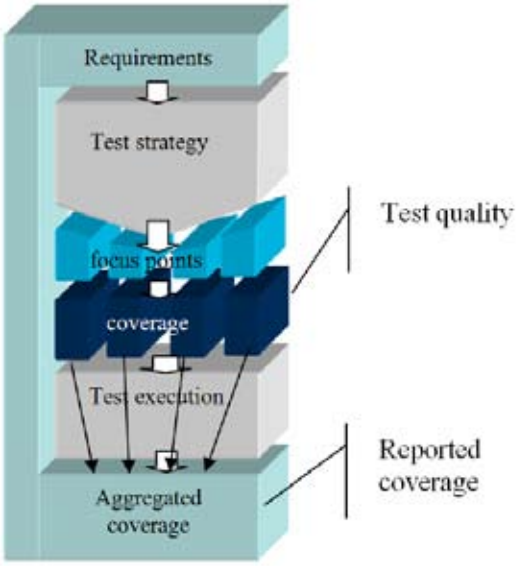


wishes and requirements towards the product, but also accepts the fact that "testing everything is impossible". Therefore we need to know what aspects of the software system are really necessary to undergo a test. These strategic aspects need to be translated into measurable focus. Focus like: "Calculations need to be correct for all combinations of pricing-aspects in the billing module" or the user-process of the Call Centre needs to run without blocking problems.

What concept can fill this gap between the focus points, translated from the customer requirements, and the tests prepared and executed by the tester? Especially in a short period of time! This concept is called 'Coverage'. Not coverage like: zero defects or service ready. The coverage needs to be related to the focus. With a good test script the tester will be able to cover 75% of the combinations of pricing aspects by his test. This is the answer to the customer's question. The customer will gain confidence if IT and Testing can explain *what* is covered. So its not about: "how much did you test?"

Customers and coverage

Based on this, we can answer the real question of the customer which is translated into the test strategy. We are also able to report accordingly. But all of a sudden the customer is not interested in this information. Even if we split it up into focus points or different perspectives, even if we link the coverage one-on-one to these focus points, the customer appears not to be interested. It is often too detailed and does not answer the real question: "does the system work for me?"



What we discover here is the difference between having confidence in the testing department (IT) or process, and having confidence in the quality of the software. Up to this moment we translated the customer's requirements into a strategy, but at the end of the day, the only 'one' who is interested in this strategy is the testing process. The conclusion is that we have only discussed the coverage towards 'test quality'.

We must conclude that the customer takes this 'test quality' for granted, Testing gets paid to do high-quality work. The customer wants to have insight into which parts of his wishes and requirements are tested. The test report needs to be a high-level coverage picture: a mix of testing quality coverage aggregated to requirements coverage.

At home I tried the following on my wife. She wants to buy a bike. I asked her what her quality requirements were for that bike. She had only one single requirement: that bike needs to have the quality of a bike, with wheels, a handlebar and pedals etc. It should not break at the first hole in the road.

This is the customer; not interested whether the pedals are tested in all possible positions, the only and main interest of the customer is whether the bike rides like one can expect from a real bike.

If testing takes care of filling the gap between Strategy and Execution with a clear coverage picture, testing will be *the added value* to an organization and its customers.

Biography

In 1996 Erik started his career at an ERP company as a tester for a distribution and workflow application. Over the years he moved on to test management and test process improvement both in the Netherlands and abroad. He has gathered a lot of experience in several organizations testing ERP in amongst others insurance, utilities, and the banking industry. He has managed test projects for newly built software, maintenance projects and more technical, migration projects.

Erik gradually moved from test management to training and coaching the test managers and testers and to set up or improve the processes within these companies. Erik is an accredited trainer for ISTQB Foundation and an ISEB Practitioner as well as TMap Next Foundation and Advanced.

Erik is currently Test Consultant and Trainer at Polteq International Testing Services BV in the Netherlands.

Testing Organizations: The Stability Challenge

Dear colleagues and readers,
I am pleased to welcome you to my forecourt, and would like to invite you to take an active part in discussing subjects that will be raised in this and coming columns. Please feel free to send pros and cons, arguments and comments to the following email address: yaron@testingexperience.com and in a short time we'll publish a blog through the 'testing experience' where we all will be able to share.

The forecourt is the centre of my home. It's where I host family get-togethers and relaxed evenings with friends and colleagues. The type of place where you can find yourself sitting, relaxing, with all the time in the world, to talk, to share your thoughts and get good ideas and advice from the company. Apart from the talk, there is a great atmosphere, snakes, good food, beer, red wine and the rest.
So, my dear readers, I am inviting you to help create and take part in a virtual forecourt, to feel the atmosphere, and take part in stimulating discussions. Take a drink in your hand, sit on the sofa or your porch, stretch back, and here we go……

The topic I've choose to share with you focuses on the testing organization that we manage. It is a challenge for both the testing manager and the test engineer, and requires all our focus and attention. For those of us who are managers, it is also a part of our professionalism.
We find ourselves investing a lot of effort, time and money, in setting the vision and mission for our testing organization, coaching, training, and creating it, shaping it, strengthening it, and making it ready for current and future activities. We have reached the point that our organization is stable, strong, functioning according to management expectations, and giving added value to the company. We have interesting jobs, and a positive atmosphere.
Well folks, if you thought that now is the time to rest, than you're naïve. On the contrary, you need to continue to assess your organization, the way it functions and the way it is positioned. In addition, you need to be ready for any surprise, meaning that you need to continuously check the heartbeat of your organization.

In our high-tech industry, things move fast, knowledge is one of our biggest assets, and it is possessed by our employees. Therefore your staff is your biggest asset, and you as a manager are responsible for both the people and the knowledge. You must continuously maintain these assets, leverage them and take care not to lose them.

So far I probably haven't said anything that you don't already know. However, putting it into practice in your daily life is often far from obvious. In the outside world there are opportunities, there is excitement, there are new challenges and new horizons for testing professionals. This is the battle you are fighting, and which you have to win.
The problem is that you have to work out how to win, otherwise all your hard work, and the work of your engineers, is lost.

So, what can we do? Well, I have few tips for you to take to the road. If you are lucky, some of them are already part of the culture of your company and organization. If not, you will need to be pro-active in deploying these tips, remembering that they need to be managed as a process or project.

I'm going to start with the first two important tips that serve to 'align expectations' (an approach that I'm a fan of): Role/Job descriptions, and Career paths.

Since the page is almost ended, my next column will discuss these two tips in depth.

Taking you back to my invitation to stimulate your minds, let me know what you think, get involved, and I'll see you next time.

Biography

Yaron Tsubery has been a Director of QA & Testing Manager at Comverse for seven years. Yaron is in charge of a large group that includes Testing Team Leaders and Test Engineers. The group deals with Functional, Non-functional, Load & Performance tests, some of which are done off-shore. The group has much experience in managing and controlling acceptance tests. Yaron is also known as a manager who has built testing groups in his past and also acted as a consultant to testing managers while they were building their testing groups.
Yaron has been working in software since 1990, and has more then 15 years in testing field as a Test Engineer, Testing TL, and Testing Manager. His original profession was Systems Analyst.
Yaron worked in Product Management, Project Management and Development for at least 3 years before becoming a Director of QA & Testing.
Yaron is a member of ISTQB (International Testing Qualifications Board – www.istqb.org) and is the President and founder of the ITCB (Israeli Testing Certification Board – www.itcb.org.il). He is a member of IQAMF (Israeli QA Managers Forum) and in SIGiST Israel.
Yaron has been invited as a speaker to some important international conferences to lecture on subjects related to testing.
Yaron has lots of experience in banking business processes. For the last 7 years he has implemented best practices in a field where he is in charge of producing complex systems for telecommunication companies such as Vodafone, T-Mobile, NTT DoCoMo, Verizon Wireless etc';
Main systems in Mobile technology: i-mode, Mobile GW and Billing Proxy, in SI projects: Content Data Management solutions, Video Solutions, IPTV and IMS.

Pragmatic Software Testing

"IF WE WOULD NOT LISTEN TO THE MARKET AND TO THE BUSINESS ASKING US TO IMPROVE OUR TESTING KNOWLEDGE AND OUR AUTOMATION, WE SHALL BE ALL OUTSOURCED VERY SOON"

Software Testing has become extremely complex. The test engineer and test leader are required to have knowledge far beyond what they have today. In a bundle of high selection of courses, you will have a chance to discuss advanced topics and automation challenges of testing today. We present you with advanced level pragmatic course in testing as well as certification courses from ISTQB-CTAL both for the Test Manager and for the Test Analyst, plus a whole new set of Automation professional career tracks from our QTP family. More courses are available.

As knowledge gets accumulated at SELA's consultants who are the trainers, we provide professional testing services and consulting as well in the areas of risk based testing, test management, defect management, team building, test process improvement and more.

Join other testing professionals for a discussion on pragmatic advanced testing topics and various QTP automation challenges of software testing.

Get the most pragmatic view over advanced day-to-day challenges and issues in software testing! We shall discuss real life scenarios, think of possible solutions, analyze case studies where relevant, and have a practical hands-on with all of our QTP Automation track courses.

For a complete schedule visit our Web site ▸▸▸

- Pragmatic Advanced Testing
- ISTQB-CTAL: Test Manager
- ISTQB-CTAL: Test Analyst
- Automation Professional Career tracks:
  - Discovering QTP (first-steppers)
  - QTP.NET & .NET-Web Programmer
  - QTP Web Programmer
  - QTP SAP & SAP-Web Programmer
  - QTP Data Oriented Programmer
  - QTP Terminal Emulation Programmer
  - QTP Multi-lingual Programmer
  - QTP Java Programmer
- Professional Testing Consulting Services

SELA COLLEGE

Sela College Israel

Phone: +972-3-6176066

Fax: +972-3-6176605

www.sela.co.il

Sela Canada

Phone: +1-416-628-3841

Fax: +1-416-628-3801

www.sela.co.il/canada

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

ISTQB
Certified Tester?
Get a FREE
Version

CaseMaker®

www.casemaker.eu