June 2011

testing experience

The Magazine for Professional Testers

printed in Germany

print version 8,00 €

free digital version

www.testingexperience.com

ISSN 1866-5705

Improving the Test Process

Certified Agile Tester

Pragmatic, Soft Skills Focused, Industry Supported

CAT is no ordinary certification, but a professional journey into the world of Agile. As with any voyage you have to take the first step. You may have some experience with Agile from your current or previous employment or you may be venturing out into the unknown. Either way CAT has been specifically designed to partner and guide you through all aspects of your tour.

The focus of the course is to look at how you the tester can make a valuable contribution to these activities even if they are not currently your core abilities. This course assumes that you already know how to be a tester, understand the fundamental testing techniques and testing practices, leading you to transition into an Agile team.

The certification does not simply promote absorption of the theory through academic mediums but encourages you to experiment, in the safe environment of the classroom, through the extensive discussion forums and daily practicals. Over 50% of the initial course is based around practical application of the techniques and methods that you learn, focused on building the skills you already have as a tester. This then prepares you, on returning to your employer, to be Agile.

The transition into a Professional Agile Tester team member culminates with on the job assessments, demonstrated abilities in Agile expertise through such forums as presentations at conferences or Special Interest groups and interviews.

Did this CATch your eye? If so, please contact us for more details!

Book your training with Díaz & Hilterscheid!

Open seminars:

July 18 - 22, 2011 (Summercamp) in Berlin Germany
August 15 - 19, 2011 in Berlin, Germany
August 15 - 19, 2011 in Helsinki, Finland
September 26 - 30, 2011 in Mödling, Austria
October 10 - 14, 2011 in Berlin, Germany
November 07 - 11, 2011 in Mödling, Austria
December 5 - 9, 2011 in Berlin, Germany

Díaz & Hilterscheid GmbH / Kurfürstendamm 179 / 10707 Berlin / Germany
Tel: +49 30 747628-0 / Fax: +49 30 747628-99
www.diazhilterscheid.de training@diazhilterscheid.de

Díaz Hilterscheid

Editorial

Dear readers,

the summer is a step away from us, and the time comes for relaxing a little bit and also for planning ahead how the fall is going to be.

We received a lot of papers and decided to again take a combination of new and old authors, giving also new authors the chance to write about their thoughts, experiences and opinions.

I'm happy with the result and hope you like it.

We just closed the Testing & Finance in Germany. It was quite successful, and we did have very nice keynotes and talks.

We announced that the next Testing & Finance will take place in London, and that May would be a nice time. We will announce the call for proposals quite soon.

The call for proposals for the Belgium Testing Days is closed, and we received around 100! A lot of work for Mieke Gevers & Co.

The proposals look very promising. We will have a very good conference. Don't miss it. Please save the dates.

We have started an Agile Tester Award. We would like the community to choose the "Most Influential Agile Testing Professional Person" in 2011. Please go to the website and see the terms and propose your favorite professional.

I spent some days over Easter with my kids in Heiligendamm. A very nice place. We had "Osterfeuer" at the beach. It was very, very nice.

I wanted to ride horses, bicycle and walk around with the kids at the Baltic see. I did my planning without the customers (the kids). They just wanted to be in the pool the whole day, so we did nothing else apart from swimming and eating lunch at the pool. It was wonderful, but after a few days of six hours a day in the pool, my old skin started to suffer... ;--)

If you have not been there, try it out. It is nice.

I wish you enjoyable reading and will be happy to hear your comments.

Contents

Illusions and misunderstandings about software testing

by Hans Schaefer

This article is intended to help you argue for the need to check and test software. Many stakeholders involved in the software industry have illusions about testing that hinder you to do a good job. There are management illusions, developer illusions, tester illusions and user illusions.

The most common are:

- Management illusions
  - What is testing, at all
  - Anybody can test
  - You can test in quality
  - Some products need no testing
  - Automated testing vs. exploratory testing
- Developer illusions
  - My software has no bugs (or too few to care)
  - The testers are going to find all bugs anyway
  - Thorough unit testing is enough
- Tester illusions
  - I am allowed to stop bad software
  - I need to test everything
  - I can test everything
  - If it works, it is good enough
  - Thorough system testing is enough
- Customer / User illusions
  - Buy it and forget it
  - We do not need to test
  - Testers do not need to communicate with us

Some of these illusions have been argued against for a long time, and it should be known that they are wrong. However, as testers, we still run into them and this creates trouble in our work. The final illusion is that the illusions mentioned here are the only ones. This is not true. There are many more of them.

Management illusions

What is testing

Testing is misunderstood, and because of this misunderstanding, people may think very differently.

When software was new, people thought testing was an activity that would show that software worked. Testing was demonstration, it was positive. Running the program to show that it is right.

This is, to say the least, dangerous. It is theoretically and practically impossible to show that a program is correct. Additionally, what is right for one user may be wrong for another, depending on personal needs. We have to distinguish between "technical quality", where the testing task is "verification" and "user quality", where testing does "validation". Psychologically, it is also problematic: If people shall show that things are correct, they will do their best to do this and will tend to overlook problems that do not contribute to this goal.

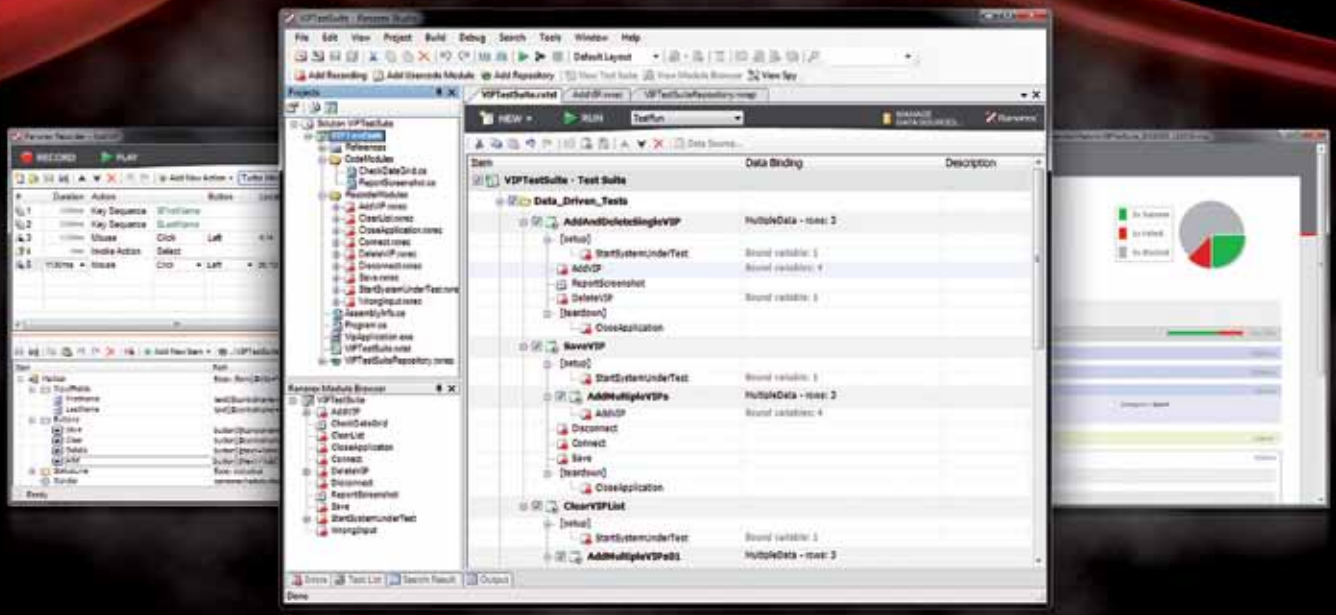
As a reaction to this, testing was defined as the opposite: Testing is trying to show that the software does NOT work. The slogan "You make it, I'll break it!" is psychologically a much better definition. We try our best to find trouble. Only if we do not find trouble, it is a side-effect that our trust in the software product will increase. This definition also conforms to the usual way scientific advances are criticized and then accepted: New theories are finally accepted after comprehensive criticism. The trouble with this definition, however, is a kind of war between testing and developing. This is detrimental to productivity.

Testing was then redefined: Testing is a measurement task: Measuring product quality and/or measuring product risk. Thus, testing is independent of development, tries to be objective and just collect information about the product for whoever is interested. This sounds better. However, there may be a motivational problem: If testers find no trouble, they may see their work as boring. If product owners just get positive feedback, i.e. no trouble found, then they may see test resources as a waste. On the other hand, it is not often true that testers find no trouble, and there is value of information. Actually, one might say: Testing is the KGB of the product owner. Its task is to provide information.

Ranorex®

Automate your User Interface Testing

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



Record and Edit Reliable Test Actions

Manage and Execute Your Automated Tests

Reproduce Bugs and Maintain Your Tests

- ✓ Use connectors for data-driven tests
- ✓ Build robust test automation frameworks
- ✓ Generate EXEs for pure flexibility
- ✓ Write custom code in C# or VB.NET

Award-winning test automation tools, which allow testing of many different application types, including: Web 2.0, WPF, Flash/Flex, Silverlight, Qt, .NET, 3rd Party Controls, and Java.

Download your 30-day Trial at www.ranorex.com

However, most people do not like the KGB.

Thus, testing needs another changed definition: Testing is any activity that evaluates a product in order to provide information, feedback and contribute to future improvement.

This subsumes that testing is not only pounding at the keyboard and looking at the screen (difficult to do with most control programs anyway), but that testing is any evaluation done on any part of a product. It may be reading or reviewing documents. It may be static analysis using tools (like spell checking documents or analyzing code), or finally, it may be executing the product with inputs and checking outcomes. The goal of it all is to provide information to product owners, but also to provide feedback about what is done wrong, and helping people who are doing this to improve. Thus, testing is feedback to help learning. Every problem found can be analyzed in order to find more such problems in the future (testing improvement), or in order to prevent occurrence of such problems in the future (development improvement). The basis for finding problems, however, is a critical attitude with any tester: "It is best I look for problems".

Many illusions about testing are prevented when we agree on a common definition.

Anybody can test

This essentially means that testers are second class citizens. Whoever is less educated or less skillful, can be used to test software. The trouble is that testers need to be highly skilled and knowledgeable, but in very different domains: Test techniques, using test tools, test automation, program development (in order to know what could be wrong but also in order to program test scripts) and product domain. A knowledgeable and skillful tester will find more bugs more efficiently. Good people save money.

You can test in quality

This illusion is as old as product development. In the hardware industry they have learnt this, the hard way. With software we still have to learn it. Bugs cost more the later they are found. When fixing bugs there is the risk of introducing new bugs and side-effects. This increases with the size and age of the product. Finding bugs by late test execution is not very productive and dangerous. If testing reveals serious unreliability, it may be too late to fix the bugs. Bugs should be found reviewing specifications and design, not executing test cases. If something is unreliable, it should rather be thrown away and built anew from scratch. Otherwise, bad solutions are carried on and will cost too much later.

Finally, if the solution is all wrong, no fixing of symptoms after a test will help anyway.

However, there is a chance to IMPROVE quality if the product entering test is not too bad! If we have a product that is reasonable, we may shake out some more bugs by late exploratory testing and this way improve quality. And we should do so. We do not find every bug by systematic testing, automated testing or reviews or static analysis. Some are only found by dynamic testing using peoples' brains. ("Brain-in-the-loop" testing.) However, using this kind of testing as the only means is normally a waste of time.

Some products need no testing

A recent trend is releasing software with faults. In many of the most popular internet services, the users find the problems and these are fixed in production use. However, people do not die from bugs in such products. And still, there is a lot of testing in such products before release.

Human work is error-prone. Whatever humans do, there is the effect of human error, thus if something is too dangerous in use, there must be testing. Testing actually means using any technique there is in order to find information on possible defects. Part of this is reviewing, part is using automatic checking or analysis tools, part of it is designing and running test cases. If none of these methods is applied, it is unknown if anything could be wrong and the resulting risk is, in most cases, far too high.

Thus, what is testing?

Testing is using any analysis technique in order to get information about the test object. Testing is measuring the risk of using the test object. The tester is the product owner's KGB. However, testing is also learning: Learning how to prevent bugs and how to find them earlier and more completely. This way, testing contributes to continuous improvement.

Automated testing vs. exploratory testing

Actually, there are two illusions:

1. We have so much automated testing, thus we do not need manual testing, and
2. Exploratory testing, mainly done manually, finds so many bugs, and therefore automated testing is a waste of resources.

Automated testing IS important. It gives us the possibility to test the main features of software for regressions, as well as try the software with different settings and platforms. Without automated testing, it is definitely not safe to try changing anything. With ever increasing complexity and ever increasing possibilities to introduce bugs, we have to use every means of fighting them. Automated testing is one of our chances to do so.

However, automated testing only checks what it is told to check. Telling it means programming. A tester has to predict what is important to check. Many features are necessarily left out. Additionally, there is a concern for maintainability of automated test cases and scripts. This often leads to relatively simple tests, testing one feature at a time, and to leaving the user interface out of the loop, testing through APIs. Manual exploratory testing, on the other hand, keeps the brain in the loop. The human brain is adaptable and human testers will check many things an automated script will not check. Additionally, exploratory tests may check many features at a time, touring through the applications in different ways, simulating real user actions. However, exploratory testing becomes a hopeless undertaking if the product is too unstable. Thus, both techniques need to be applied.

Developer illusions

My software has no bugs (or too few to care)

This corresponds to the management belief that some products do not need testing. Humans err, developers err. Developers need the other pair of eyes a tester would give. A typical developer introduces about 1 bug for every five lines of code. Even the best ones are only a factor of 20 better. This would be 1 bug in 100 lines of code. That bug may be fatal! If it is not considered worth caring about, what does the user think about the product?

Users care if the software does not work. Developers believing in their own infallibility need especially thorough reviewing.

The testers are going to find all bugs anyway

No. According to industry statistics collected by Capers Jones, Productivity Research, no testing method short of high volume beta testing can be relied on to find more than 50% of the bugs. Even with six phases of the best testing, 3% of the bugs will then still survive. And testing is like a filter: Garbage in leads to garbage out. The more bugs in, the more bugs out. Bugs also take time from testing. The more bugs are found, the more work is used to isolate bugs, document them, fix them and retest and regression test. Thus less testing is done, and less of the remaining bugs are found. Believing in the testers is dangerous risk compensation. Introducing start criteria to testing may be a solution.

Thorough unit testing is enough

In Agile methods, a lot of automated unit and continuous integration testing may be done. Many people believe this is enough. However, this testing is directed against coding and interface bugs. The total system may still not meet the requirements.

Every test level has its own special objectives. Good unit testing reduces the bug content in the system, but testing the parts does not guarantee that the whole is good. (Anyway, testing cannot guarantee anything anyway).

Tester illusions

I am allowed to stop bad software

Beware of being the quality police! Releasing software is a plain business decision, to be made by the product owner. A tester's task is to provide information, to measure the risk of releasing. The decision to release or not depends on the risk, but also on factors that are beyond our control, like the state of the market. Sometimes products full of bugs may be released because otherwise a market window would disappear. In that case it may be better to get product sales going and fix the problems later, even if that might be expensive.

Testers may, however, show possibilities for action. Such possibilities should be used during the "end game", right before release. For example, if some features are full of serious bugs, they might be removed or disabled, or simplified. If they cannot be cut out, the testers may propose to postpone release. However, testers do not decide on release. A test report is merely a statement about the risk to release.

I need to test everything

Testing everything is impossible. This is exhaustive testing. Testing everything would mean to test every combination of inputs, environment, timing and states, which creates an astronomical amount of test cases. Testing a program in a 32-bit architecture with only one integer input would require 232 test cases.

However, testers may test "everything" once. This may be testing every feature, every requirement, every window, dialog box, field, batch job, background activity or whatever is visible. Testers may continue combining this, first pair wise, then further on, as far as they consider necessary. However, any kind of combination testing will explode the number of test cases and thus be expensive.

A test will always be a very small sample with regard to the total possibilities, and bugs may go unnoticed. Statistically speaking, that sample is not even valid, but it is the best we can do.

I can test everything

This is plainly not true. See the details described for the previous illusion. If a tester says this, he/she is either ignorant or the program is trivial. Even with automated tests this can never be achieved.

If it works, it is good enough

Well, it depends. If the purpose is a product demonstration at a conference, with full control by the demonstrator, then this may be true. If a product shall be released to real users, this is never true. Users do all sorts of things. They seldom behave the same way as we think. They do things in different orders, repeat things, leave out things and do many things wrong. All this needs to be checked out by testers. Even if we run extensive automated tests, designed using all the best techniques available, exploratory testing will in most cases find some new bugs, just because we introduce new variations. Users would do this. Thus, testing never gives full safety.

Thorough system testing is good enough

System testing is testing the whole integrated system. It is checking that the system conforms according to requirements. The trouble is in the details: Testing every requirement one or a few times will probably leave many parts of the code outside of testing. The parts typically left out unknowingly by system testers are error and exception handling. As developers tend to introduce more bugs in these parts than in the mainstream code, these parts are especially error-prone and should thus be tested more. This is often impossible in system testing.

Many testers who have tested systems where unit and integration tests were badly or not done, can report on problems. It is often frustrating to test a system which is not robust and where trivial bugs show up. As a tester, one gets the feeling of being misused to clean up other peoples' mess.

Customer / User illusions

Buy it and forget it

This is a dangerous one! Some users think developers know what to do. They also think they themselves do not know how to acceptance test and let the developing organization do this task. The result is often a disaster: The system does something, but not what the customer intended to have. Developers will implement what they think the customer wanted, not what the customer REALLY wanted. ("The inmates run the asylum.")

Modern methods all emphasize that the customer or a customer-like person must be part of the project work all the time, in order to specify what they want, in order to see what happens, in order to be available if developers ask, and in order to make sure that there are reasonable acceptance test cases. If a customer chooses not to be available, development is, to say the least, a high risk.

Customers need to be available and are responsible to design acceptance test scenarios.

We do not need to test

Why? Because the software will do what we need anyway? If a product is expensive enough, it is better to check first. The need to test only disappears with very cheap standard products where we can base our decision on the decision many others have made

before, and on their experience. Otherwise we have to test. This is a variation of the "buy it and forget it" illusion.

A variation of this, from Berit Hatten: Customer: We do not need to check how it works at our place / in our organization... etc. It worked with a friend of mine...

Would this work on YOUR platform? With YOUR use?

And another one: Customer manager: Eh, do you need to test? Do you sell bad software? – I think the best answer is not to sell to such customers. Or, as an alternative, after very careful explanation of the above illusion.

Testers do not need to communicate with us

This means the testers will not know what we, the customers, want. This also means that we do not get the knowledge of testers about what could be interesting to have a look at. If it is not important that a product is reliable and meets expectations, then communication with testers is not important. Otherwise it is. There should be a lot of communication between customer, tester and developer.

References

There are definitely more illusions than the ones mentioned here. There are also many more references where illusions may be found, as well as advice what to do about them. The following, however, were the main inspirations for this article.

(1) Gerald Weinberg, Perfect Software and other illusion about testing, Dorset House, 2008. This book is inspiring. The author describes many more illusions and fallacies and how to fight them. He also describes much of the psychological reasons behind.

(2) James A. Whittaker, Exploratory Software Testing, Addison-Wesley 2010. This is a great book about testing using improvisation and peoples' brain. It also does away with the illusion that no other testing than this is necessary.

(3) Lisa Crispin and Tip House, Testing Extreme Programming, Addison-Wesley 2003. This book is rather technical, showing how to implement automated unit testing. However, it emphasizes that testers are not "vacuum cleaners", that they have a right to a minimum quality in what they have to test.

(4) The Norwegian government's treatment of Maria Amelie in January 2011 as a key example of counterproductive bureaucratic management practice.

> biography

Hans Schaefer

57 years old. Specialist in software testing

Independent consultant in software testing and testing improvement matters. Guest lectures at several universities in Norway about Quality Assurance and Software Testing. Public and inhouse seminars in Software Review and Testing in Scandinavian and European countries. Regularly speaking at conferences. Several best paper awards. (Best presentation award at CONQUEST 2003, best paper in 2004).

- *Test coordinator, ISEB Certified Tester (Foundation Level), ISTQB Certified Tester (Foundation and Advanced Levels) Chairman of Norwegian Testing Board.*
- *Active participant in running museum trains in Norway, certified for safety critical services on steam locomotives.*
- *M. Eng. from Technical University of Braunschweig, Germany.*
- *Computer science, railway signaling.*
- *Development of real time process control software at the Fraunhofer-Institute in Karlsruhe, Germany.*
- *Work with Center for Industrial Research in Oslo, Norway.*
- *Development of parts of an IDE-tool (Systemator). Later developing test tools and leader of quality improvement program. Consulting and lecturing about software quality.*

How do you become an Agile tester?

by Sandra Güttig, Martin Großmann & Heiner Grottendieck

What brought us to CAT?

The independent Test and Integration Center (TIC) of T-Systems Multimedia Solutions GmbH (T Systems MMS) with its over 100 ISTQB-certified test experts and 25 security specialists works to ensure the quality and security of web applications. Our accreditation by the Deutsche Akkreditierungsstelle has made us an officially recognized software testing laboratory for the Internet and multimedia industry in Germany since 2003. In the past we have therefore mainly followed traditional software development procedures.

As our TIC performs some of the testing services for the software developing business units of T Systems MMS, we are regularly inspired from within our own company to explore the latest trends and technologies. Therefore we testers were also involved from the very start when coaching on Agile methods using Scrum first began in early 2009. Our human resources department organized a cross-business-unit coaching programme for two large projects and booked Jutta Eckstein, a recognized expert in Agile methods, to deliver it.

Numerous projects and programs have since been successfully completed within our company following the Agile method. Of course there were also difficulties and disappointments. These practical experiences allowed us to clearly recognize that things are very different when testing in Agile projects compared to

Fig. 1: Training plan (source: http://www.agile-tester.org/syllabus.html)

following the ISTQB textbook. We therefore sought ways to specifically increase our Agile testing know-how. An initial step was taken in the spring of 2010 in the form of coaching provided by a consultant from Díaz & Hilterscheid (D&H) (project interviews and action recommendations), followed by a lecture for the entire TIC and a brief Scrum crash course. However, we were somehow still missing targeted instructions from the testing perspective.

Therefore the suggestion from our training partner D&H came at just the right time, and we participated in the preparation for the "Certified Agile Tester" (CAT) programme. The aim of the CAT programme is to close the gap between ISTQB and Agile development. In October 2010, a colleague from the management team took part in the first CAT pilot day in Berlin, which ran through day one of the CAT programme. It was clearly apparent that this was an ambitious programme with a large proportion of practical exercises. In January 2011, a tester from the TIC took part in a full pilot run in Potsdam. He gave us the green light for our first live training course. Together with D&H we prepared the internationally first training course and certification. From 28 February to 4 March 2011, ten testers from T Systems MMS in Dresden took the course with two lecturers from D&H.

Training procedure

As a pilot course, the "Certified Agile Tester" training course was held with its actual target group for the first time. Most participants were experienced testers who had previously worked with more traditional testing methods and otherwise had little or no experience in the Agile environment. Two colleagues who were already familiar with the Agile method rounded off the group.

The training plan is shown in Figure 1. The course and the accompanying exam are currently offered only in English. The fact that the lecturers taught in English acquainted us with the specific vocabulary, expressions and required terms, allowing us to better prepare for the written part of the examination.

As additional exam preparation we were given exercises every day which we could work through at home. The following day we jointly reviewed the solutions and discussed some of them extensively. This also gave us an indication of the type of questions we would have to deal with and an idea of what points were considered particularly important in developing a solution.

For many of us, a course held in a foreign language was a rather unusual way to learn and engage in discussion. However, we were also expressly invited to switch to German if we were uncertain, so that the discussion would not be limited due solely to language constraints.

The course was taught by lecturers Werner Lieblang and Heiko Köppen of D&H. One lecturer was responsible for presenting the course content, the other for the practical exercises; they complemented each other very well.

The practical exercises allowed us to immediately apply what we had just discussed. For this purpose our group was divided into three teams which completed the respective exercises together.

Fig. 2: During the CAT exam

Planning iterations, that is, recurring procedures, and estimating required effort were practiced by constructing buildings and vehicles using building blocks. A second exercise focused on testing. We accompanied the virtual development of a web application through several iterations, planning and carrying out the necessary tests at each point. The objective was also to find discrepancies from the software specifications.

The aim of also familiarizing those participants with the Agile processes and methods who had previously worked only in traditional environments was fully met. Open questions were clarified in discussion sessions, so that by the end absolutely every participant had understood the Agile method of thinking. In future, however, more time should be set aside for this, as participants' prior knowledge varied very widely. This was in contrast to the rather extensive syllabus, which had to be worked through in four days and allowed little leeway.

Due to the course's pilot status certain aspects were not yet running entirely smoothly. Particularly the use of technology for an exercise proved more difficult than expected because the required software was not immediately available. The lecturers handled any difficulties that occurred very well and showed great dedication in ensuring the successful outcome of the course.

The examination

A distinctive feature of the Certified Agile Tester training course was the three-part assessment. Throughout the week we were observed in how we worked together in our teams and whether we were capable of taking responsibility for certain tasks ("soft skills assessment"). This section was evaluated by the two lecturers.

The examination day itself was divided into a practical and a theoretical section. The exam was administered by an employee of iSQI GmbH. First, the participants had to plan and execute a test over several iterations based on a web application. It was particularly important to document each step in order to make it possible to assess participants' performance.

The following theoretical section required us to answer questions in sometimes very extensive text. No multiple-choice questions were asked.

For many of us this type of examination was a new experience, which also gave rise to some insecurities beforehand. Having to answer the questions entirely in English also required a greater amount of time, which should not be underestimated.

The use of technology during an examination poses a certain risk, as specific factors such as problems with the application or the hardware cannot be influenced by the examinee and may therefore negatively affect the outcome. It must also be ensured beforehand that any required devices are available and suitable for carrying out the examination. For example, booting from external media such as USB sticks is often not permitted in corporate environments. Our colleagues at T Systems MMS were, however, able to unbureaucratically provide assistance by lending us brand new notebooks.

> biography

Sandra Güttig
holds a Diplom in Media Computer Science, is ISTQB- and CAT-certified, and has many years of experience as a software developer and tester in various traditional development projects. She had been working as a tester on an Agile project of T Systems MMS for over one and a half years and has come to value the advantages of Scrum. Her core responsibilities are in the field of test automation.

Martin Großmann
holds a Diplom in Business Informatics (BA), is ISTQB- and CAT-certified, and has been working as a test designer at T Systems MMS for several years. As a test automation specialist he has worked on several Agile projects and thus gained a lot of experience in this environment. In addition to project work he manages the development of the automation framework for the Test and Integration Center of T Systems MMS.

Heiner Grottendieck
holds a Diplom in Communications Engineering, is IREB/CPRE certified, and has been working in industry hardware and software development projects for 18 years. For four years he has been managing the Technical Test department of the T Systems MMS Test and Integration Center, where in addition to load and performance tests and application monitoring, test automation plays an important role in numerous customer projects. He has been working with Agile testing in this context since 2009.

Plan, do, test and act

by Bert Wijgers

Testing is an activity that can lead to improvement of the software creation process. We have to go one level deeper in order to learn how to improve the software testing process. We can test the products of the testing process. It is difficult to assess process quality with metrics, but we can assess the craftsmanship of individual testers. This is done by testing them.

An obvious starting point for any quality improvement activity is what is commonly known as the Deming circle: plan, do, check and act. First you make an improvement plan and then you execute it. Then you check whether the plan has worked. If it didn't, you make a new plan. If the plan worked more or less, you act, that is, you make adjustments to make it work better. If the plan worked really well in the first place, you can start to make a new plan to improve even further. Whichever way, the circle starts again at some point and this should continue indefinitely.

Deming always referred to this circle of quality improvement activities as the Shewhart circle. Deming (1986) made only one modification; he replaced "check" with "study". Unfortunately, this modification did not catch on, probably because "plan, do, study and act" doesn't sound as good as "plan, do, check and act". Therefore I propose an alternative that is in line with Deming's idea that the third activity is much more than just checking and that has the same rhythm and rhyme as Shewhart's circle: plan, do, test and act.

Software testing is part of software creation. In agile projects this is made explicit; testers work in multidisciplinary teams with developers and designers. In waterfall projects test teams seem to have a more independent position, but still they are part of the bigger picture. Software cannot be built without a plan or at least an idea, it cannot be tested before it has been built, and it cannot be released before it has been tested and defects are solved. Even in test driven development, where the tests are designed before the software is built, eventually the software has to run in order to pass the test. When testing, building and designing become intimately interwoven, the quality improvement circles become shorter. However, the same activities remain; plan, do, test and act.

The quality improvement circle can be identified at different levels. So, if we want to improve the test process, we have to make a plan and do it. After that, we test and, if necessary, we act upon the test results. The approach is always the same, but it can be applied at a different levels.

Circles within circles

Dynamic testing, that is, testing software by executing it, is part of the software quality improvement circle. First a plan is made that we can call a functional design; in fact, any kind of formal or informal documentation that is meant to guide developers can be considered as a plan. Based upon this documentation the software is built; the plan is executed. Then it is our turn.

Part of dynamic testing is checking. Whenever we do tests to confirm that information systems behave according to specifications, we are checking. But testing is more; another part of testing is non-confirmative. These are the kind of tests that are meant to find defects. This is a very different goal that requires different means. Testers who do non-confirmative testing need not only be good observers, they also need to be imaginative. Both kinds of testing, confirmative and non-confirmative, are valuable and they complement each other.

Static testing, in the form of documentation reviews, is part of the documentation quality improvement circle. Since documentation is used as the plan of the software quality improvement circle, reviewing it is a very cost effective type of testing.

In short, testing is the critical examination of products from the design activities and the building activities. If we happen to find a defect, we dive into it to pinpoint the cause and to estimate the effects. Through testing we contribute to the quality of the software only indirectly; the defects we find will be fixed, which makes the software or the documentation better.

Products and preconditions

Products from the testing activities have to be critically examined as well. These products include test scripts and issue reports. Together these products give an idea about the quality of the products under test. To assess the quality of test scripts and issue reports, we have to turn to the users of these products. Product

quality only manifests itself when a person interacts with the product. Test scripts are used by testers, issue reports are used by developers and designers.

When the products are up to the expectations of the users or when the goals of the user can be achieved, quality is perceived. Even when a product is well made, the quality can be perceived as low, for example, when the product is used by someone who expects something different or by someone who chose the wrong tool to achieve his goals. On the other hand, when a product is not well made, high quality will never be perceived, no matter which person uses the product. Testing is a way of finding out whether a product is well made. The quality of a product can only be judged by the actual users, when they use it.

One way to assess the quality of a process is to assess the quality of its products, but we also have to keep an eye on the clock. Since testing can never be complete, the optimal subset of all combinations of inputs and circumstances has to be tested. An important part of the quality of the testing process is in the choices that are made. Unfortunately, these choices can only be judged after the software is extensively used in the production environment. Defects in the production environment are input for test process improvement activities.

Apart from a sufficient amount of time, a tester, like any other professional, needs good materials. One of the things testers need to do their job is knowledge of the software and its use. This knowledge is transferred by means of formal and informal communication. More often than not this communication is documented to some extent. The quality of this documentation is very important for the quality of the testing process. It is good practice to review the documentation that goes with the software. Since documentation is never complete, all other forms of communication about the software and its intended use are important as well.

Another thing that testers need is a good set of tools. Management is responsible for a well equipped test environment. Without a good test environment the quality of the test process will falter. The most important testing tool, however, is the mind of the tester. A tester's mind needs to be challenged on a daily basis, otherwise it may turn blunt. The negative impact of repetitive tasks on the sharpness of a tester's mind should not be underestimated. Tooling can be used to free testers from this burden.

People over processes

There are big differences between testers with regard to efficiency and effectiveness. Given the same test object and documentation they will produce very different test scripts and issue reports. In order to guarantee the quality of the testing process, you want to hire the testers that find the most defects and that find the most important defects first. This can be done quite simply by testing them. Give them a piece of working software and some sort of documentation to go with it and see what happens. Make sure you know this particular piece of software and the defects it contains very well. If you need a tester to do mainly confirmative testing, you should look for good reading and observational skills.

Once testers are part of the team, they will not be judged and rewarded by the number of defects they discover. This would only motivate them to go for easy bugs and to log variations of the same bug separately. Measuring the performance of individuals and teams is a tricky business because you may end up with beautiful numbers and ugly results. When testers are too busy with their bug counts, they will not undertake any process improvement activities.

Bug counts can be used in coaching sessions, but only to help testers to become better at their job. To make clear that bug counts have nothing to do with rewards, it is advisable to coach testers on the basis of their performance in a controlled setting that is not part of their daily working space. Testers should be tested at regular intervals and make improvement plans based upon their performance. Another way to improve the performance of individuals is to let them work in pairs. In doing so, they will learn to reflect on their way of working.

Individual professional growth and test process improvement require roughly the same activities: plan, do, test and act. One thing to keep in mind, though, is that better is not always good. Testers need some freedom to follow and develop their intuition. In doing so, they will inevitably make mistakes. As long as testers are allowed to make mistakes and learn from them, the test process as a whole can improve as well.

Reference

Deming, W.E. (1986), Out of the crisis, MIT Press, Cambridge MA, USA.

> biography

Bert Wijgers
is a test consultant with Squerist, a service company in the Netherlands. Squerist focuses on software quality assurance and process optimization. Its mission is to inspire confidence through innovation.

Bert holds a university degree in experimental psychology for which he did research in the areas of human-computer interaction and ergonomic aspects of the workspace. He had worked as a teacher and trainer in different settings before he started an international company in web hosting and design for which a web generator was developed. There he got the first taste of testing and came to understand the importance of software quality assurance. Bert has a special interest for the social aspects of software development.
He uses psychological and business perspectives to complement his testing expertise.

In recent years Bert has worked for Squerist in financial and public organizations as a software tester, coordinator and consultant. He is a regular contributor to Testing Experience.

Efficient testing with Lean and TPI NEXT®

Realize cost reduction with the joint application of Lean and Test Process Improvement with TPI NEXT®

by Matthijs Naujoks, Eveline Moolenaars-Koetsier

Long test processes lots of rework and waiting, non-standardized procedures, and overlapping tests and reports. Sounds familiar? What can you do to make testing processes more professional and efficient?

Why is efficient testing necessary?

The IT marketplace is becoming more mature. Productivity, transparency, standardization and customer value become increasingly more important. IT organizations realize there is a shift in role from supplier to partner. This new partnership role means working more closely with clients - especially the business - in the whole value chain, not simply delivering services or products.

In testing, recent important aspects in contract negotiations are collaboration in off-shoring, transparency, cost reduction, quality improvement, productivity, and a higher maturity level of test processes. These aspects were normally not addressed in traditional test contracts. To realize all these new goals and to become a modern partner, a new approach to testing and test improvement is necessary. TPI NEXT® combined with Lean can help achieving this.

The test process is part of the total value chain of software development. The main cost savings can be achieved in the collaboration between all parties in the chain. This is where Lean complements TPI NEXT®. Lean seeks to maximize customer value by reducing waste in the whole value chain. In this article we first focus on TPI NEXT® and on Lean. Then we explain why the joint application works so well. The article ends with a real life example, in which the benefits of this new approach become clear.

What is TPI NEXT®?

The TPI NEXT® model gives organizations a step-by-step method for improving the test process by focusing on several key areas. This can be compared to Google Maps: to go somewhere it is first necessary to know the starting point. Based on the destination (the goals for improvement), TPI NEXT® creates a prioritization of what to improve first and what subsequently.

A strong feature of the model is that it's *business driven*. Depending on the kind of goal that has to be reached, TPI NEXT® provides a selection of key areas to focus on. Cost reduction for example can be achieved by involving testing early in the development project. Defects found earlier in the project, like unclear requirements, have lower fixing costs than defects found at the end of a project. Transparency as improvement goal on the other hand is achieved by better reporting. Reporting, however, does not necessarily contribute to cost reduction of the test process.

What is Lean?

Lean is a long-term, company-wide philosophy aimed at continuous improvement. It consists of a collection of tools, best practices and a high degree of common sense. The pitfall in Lean deployment is to focus solely on the tools and visual aspects of Lean, and not on the underlying culture and mindset. This is the most challenging and critical factor in a successful Lean implementation.

The best way to further explain Lean is by introducing Lean thinking:

- A continuous effort to eliminate waste (any activity that doesn't add value for the customer)
- A belief that there is always a better and simpler solution
- Empower employees to make the improvements happen
- The use of Lean tools and techniques throughout the whole organization.

The goal of Lean is to reduce waste, improve process speed and increase customer value. The basic principles of Lean are:

- Processes run according to flow, not in large batches
- Processes operate on pull based on customer demand
- Minimal inventory and work in process by just-in-time delivery
- Quality at the source, not by inspection afterwards
- Continuous improvement and learning

TPI NEXT® and Lean

TPI NEXT® differs from Lean in *what* to achieve and *how*. Both improvement methods are put next to each other in the table.

Method	TPI NEXT®	Lean
Theory	Focused improvement	Eliminate waste from processes
Approach	1. Determine business and/or IT improvement target 2. Assess current situation 3. Define improvements 4. Make a plan of action 5. Implement actions 6. Evaluate and redirect	1. Identify value in the eyes of the customer 2. Describe the Value Stream 3. Enable Flow and Pull 4. Empower employees 5. Repeat until perfection (never ending process)
Focus	Key areas of the test process depending on improvement target	Balanced processes in flow (instead of batches)
Principles	• Improvements aligned with business and IT targets • Alignment test process with system development process • Step-by-step improvement process	• Eliminate waste • Many small improvements are better than big leaps • Bottom-up realization and top-down support and control
Effect	Depending on business and IT target: shorter time-to-market, cost reduction, and transparency improvement	Higher customer value by faster and more efficient processes

As can be seen from the table above, TPI NEXT® is focused on key areas of the test process. It takes adjacent processes of the software development life cycle (or SDLC) into account by giving tips on how the test process and the adjacent processes can benefit from each other. However, the main focus of the improvement model is on the test process. Lean on the other hand is explicitly focused on the whole value chain of software development, up to the final customer.

Why not use only Lean? Because Lean has no experience in testing. It does not give any hints on the interdependencies of test improvement steps or on the logical order of these steps. TPI NEXT® provides exactly this kind of expertise, based on decennia of testing experience.

TPI NEXT® on the other hand, benefits from the mindset on change and improvement in Lean. An improvement process is in essence a change process, not only of procedures and templates change, but even more of attitude and mindset. This shows the advantage of the joint application of TPI NEXT® and Lean.

The improvement of a test process with TPI NEXT® consists of several steps. To maximize customer value, Lean should be put into practice parallel to TPI NEXT®. In this way the improvement process gains both from the TPI NEXT® experiences and the focus on the value chain and mindset within Lean.

Improvement process	TPI NEXT® steps	Lean steps
Generate awareness	Understand the need for test process improvement	Understand the need for test and IT process improvement
Determine goal, scope and approach	Determine target situation and key areas for chosen goal	Determine value in the eyes of the customer
Assess current situation	Assess current situation of key areas: where are we now?	Create the value stream and identify wastes
Define improvements	Determine improvements in key areas to reach the chosen goal	Define actions to reduce wastes and increase customer value
Make a plan of action	How do we get there, with whom, what timeframe, what resources?	What actions will be taken and in what order (small steps)
Implement actions	Apply the improvement actions	Implement first few actions
Evaluate and redirect	Have we reached our goals? Continue or take a new direction?	Evaluate if situation improves and correct if necessary

The real life example

The IT supplier in this example is test partner of a large company. In order to perform the test activities, the IT supplier has set up a continuous so-called testline (test factory) program containing several testlines with test processes.

The right quality of the test process in the testline is assessed on a regular basis and improved by means of TPI NEXT®. The compliant maturity level is assessed and improvement steps are taken to increase the maturity level.

To do this, the Testline Manager (TLM) fills out the TPI NEXT® questionnaire for self assessment every three months. Based on the outcomes, the Testline Manager defines improvement actions which are recorded in the continuous improvement plan. Once a year the TPI NEXT® maturity of the processes are reviewed by a TPI NEXT® consultant who elaborates on all the key areas. In this way the maturity is not only a representation of the view of the TLM, but is it also supported by an objective external audit.

In addition to the use of TPI NEXT® to improve the maturity of the test process, two Lean projects have been conducted to increase the overall efficiency of the test processes.

The results were striking: around 71% of the time spent was the result of rework and other kinds of waste and non-value-added activities.

Central to Lean is delivering customer value. To measure this the Lean philosophy defines time-related categories. These are:

- Value Added time: activities that add value to the customer. Therefore customers are willing to pay for these activities.
- Non-Value Added time: activities that don't add value or are not necessary (wastes).
- Essential Non-Value Added time: activities needed to keep the operation going, but do not directly add value.

These different types of activities can be presented in a Time Value Map. In the Time Value Map below, a 4-day timeline is presented. Based on measurements it can be concluded how much time is spent on value added activities, and how much time spent on non-value added activities potentially can be reduced.

Time Value Map

Based on measurements in the Lean projects and interviews with stakeholders, the following division in time and activities can be made:

Value Added:

These phases have direct impact on the tested system.

1. Preparation. This phase aims to have access to a test basis, and to guarantee adequate quality to design the test cases.
2. Specification. The tests are specified in this phase.
3. Execution. In this phase test cases are executed.
4. Completion. The test assignment is concluded in this phase. It offers the opportunity to learn lessons from experiences gained in the project. Furthermore, activities are executed to guarantee re-use of products.

Essential Non-Value-Added:

These phases are necessary for the test process but do not add direct value to the tested system.

1. Planning. In this phase, the test manager formulates a coherent approach that is supported by the client to adequately execute the test assignment. This is laid down in the test plan.
2. Control. The activities in the test plan are executed, monitored, and adjusted if necessary.
3. Infrastructure. The phase in which the infrastructure is set up and maintained.

Non-Value-Added:

The following activities found in the test process add no value for the customer and are not necessary to perform.

1. Re-tests
2. Multiple handovers
3. Chasing of missing or inadequate information
4. Insufficient quality test environment
5. Too many reports
6. Overlapping tests
7. Waiting time for additional information and documentation

The Lean project consequently aimed at decreasing the non-value-added activities and reducing the time spent on essential non-value-added phases to a minimum. To decrease the non-value-added activities and thus realize cost savings, several improvement actions were designed and implemented:

- Quality checks (also known as gates) were implemented between test phases to ensure quality and a smooth work flow;
- Impact analysis's were established after changes with principal stakeholders to agree on alterations in the test strategy;
- To signal quality problems in the System Test performed by other parties in the chain, a specific test, known as a Process Witness Test, was established;
- Intake on the test environment were initiated before the actual testing starts, followed by a positive release advise by the test environment manager;
- Reviews by all stakeholders in the chain (business, designers, builders and testers) were initiated: quality gate before handover to build and test.

This Lean approach gave the Testline manager and the team members of the testline a different view on the test process, which gave them a valuable mindset for improvement actions. This different view consists of a continuous effort to reduce wastes and

maximize value-added activities.

When compared to issues found in the TPI NEXT® assessments, several non-value-added activities overlap: retests, too many reports and overlapping tests. Other issues like multiple handovers (see number 2 of the 'Non-Value-Added' list above), however, were not issues according to TPI NEXT®, but the Lean project made clear that decreasing these issues meant time savings and thus cost savings for the test process. The Lean project found this issue because it looked at the test process with the end in mind, that is: the end-user instead of the test or project manager.

What are the benefits of this joint application?

As became clear from the example, Lean complements TPI NEXT® very well to further improve test processes. They both support test organizations to be the perfect partner. That means delivering:

- High customer value;
- Increased productivity;
- Low cost;
- And fast processes.

Applying TPI NEXT® gives test and IT processes an improvement boost from the perspective of several decennia of experience in the testing field. Lean supports these improvements with a boost from the perspective of process efficiency. Moreover, Lean thinking gives the organization a valuable mindset of change.

> biography

Matthijs Naujoks
is as Black Belt focused on process improvement using Lean and Six Sigma. In his approach to improve quality and efficiency he combines tools and techniques with organizational and culture change to make improvements stick. Matthijs has conducted Lean Six Sigma projects in the IT and Financial Services industry. Furthermore, he is responsible for Lean and Six Sigma courses within Sogeti.
Matthijs.Naujoks@Sogeti.nl

Eveline Moolenaars-Koetsier
is a TMAP NEXT® Certified test consultant specialized in TPI NEXT® at Sogeti. With 12 years of experience as test manager and test consultant in the testing fields of the Insurance and retail businesses, she successfully advises organizations on test process improvement. She emphasizes the strong points of an organization and experience of employees and is therefore able to advise improvements that 'work' for the organization. Eveline is responsible for the Testprov course, in which professionals learn to improve test processes with TPI NEXT® from within assignments.
Eveline.Moolenaars-Koetsier@Sogeti.nl

ISTQB Expert Level "Improving the Testing Process"

Column

by Erik van Veenendaal

As the theme of this issue of Testing Experience is "Improving the Testing Process", while being one of the authors of the new ISTQB Expert Level syllabus with exactly the same name, what else can I write about in this column? A full update and overview of this latest ISTQB syllabus, by some already referred to as the ISTQB's crown jewel, is provided in this article.

ISTQB Expert Level status

Let me first make clear to all where we are today. There seems to be a lot of confusion in the market. Yes, the ISTQB Expert Level (EL) syllabus "Improving the Testing Process" is available and released without any restrictions (to be downloaded from www.istqb.org) since October 2010. Other EL syllabi are in progress, such as Test Manager (expected autumn 2011), Test Automation (expected spring 2012) and Security Testing. To support the syllabus, various guidelines are available, such as the EL exam guideline. I personally know a number of course providers who are developing courseware in line with this new EL "Improving the Testing Process syllabus", and exam providers that are preparing for EL exams. I expect the first courses to be accredited later this year. To support the implementation of the syllabus and participants, Graham Bath and myself are writing a book aligned with the new EL syllabus. Get ready, the ISTQB Expert Level is out there!

What is an expert?

I know that many who are looking at this third level have problems with the word expert, and on-going discussions focus more on the question "what is an expert?', and not so much on the actual content being provided. First of all, this new syllabus is not targeted toward the real world-wide experts such as Martin Pol, Rex Black, Lee Copeland etc. No, it is aimed at those testing professionals that are the so-called local heroes. The person in your company that you always turn to when you have a question on test process improvement; the person that leads the organization's improvement process; the consultant who is subcontracted for challenging test improvement assessments. That is more like the target audience. ISTQB defines an expert as a person with special skills and knowledge representing mastery of a particular testing subject. [1] Being an expert means possessing and displaying special skills and knowledge derived from training and experience. A testing expert is one that has a broad understanding of testing in general, and an in-depth understanding in a specific test area, e.g., test process improvement. An in-depth understanding means sufficient knowledge of testing theory and practice to be able to influence the direction that an organization and/or project takes when creating, implementing and executing testing activities related to the specific area. It is important to emphasize that, according to the ISTQB, an expert must embody both knowledge and the necessary skills to apply that knowledge in real-life situations.

People that perpetuate the discussion on whether "expert" is the right term are, for me, those that like theory. As a practitioner, I personally prefer to focus on the content and the added value the syllabus provides. So you can decide to which group you belong? For me it is important that a third level in the certification scheme has been developed and implemented. Many (testing) certification schemes have promised three levels in the past, but never got round to it, e.g., whatever happened to the ISEB Diploma? ISTQB did!

I will leave the question whether expert is the correct term open. However, I do want to emphasize that someone who meets all exit criteria for expert level will indeed have in-depth understanding and practical skills on test process improvement. The exit criteria not only include passing the largely essay type based exam, but also having at least five years of practical testing experience in practice, and at least two years of experience in test process improvement. There is even a continuous education process defined, implying that being an expert today does not mean one is an expert for life. [2]

Syllabus content

Let's briefly look at what is being covered by the syllabus. It is not just about models such as TPI Next and TMMi, despite of what some think. Yes, these models are of importance, but there is much more to cover, learn and discuss. In fact, some say I'm an expert in test process improvement, and I believe everything (or almost everything) we need to cover on this topic is in fact covered! Together with Graham Bath, Isabel Evans and many reviewers, a top-class syllabus has been developed.

- Context of improvement; an introduction part where one learns to link test process improvement to business objectives and some fundamental concepts such as Garvin's views on quality, the Deming cycle, the EFQM framework.
- Model based improvement; a large part is spent on available models (e.g., CMMI, ISO 15504, TPI Next, TMMi). Not just by providing a theoretical overview, but learning how to apply the most important ones (see hereafter "workplace exercises"). Also the weak and strong points of the various models are discussed.
- Analytical based improvement; often forgotten but in addition to using a reference model (often resulting a top-down approach), bottom-approaches can be used and are most often highly effective and efficient. Causal analysis, inspection, defect prevention programs, defect classifications and GQM based measurement programs are all discussed in detail in this context.
- Selecting improvement approaches; it is important for the participant to be able to compare the various approaches and select those that are most beneficial for his/her organization.
- Process for improvement; much attention is provided to the improvement process. I often think this is even more important than choosing the "right" model. The process presented is based upon SEI's IDEAL process and much attention is given on how to diagnose the current situation and perform test assessments.
- Organization, roles and skills; a Test Process Group is discussed, and also how to do test improvement with remote, off-shore and outsourced teams. A large part of this section is devoted to the soft skills required for those running a test improvement program and performing test assessments.
- Managing change; test process improvement is all about changing people's behavior and therefore about change management. Although this could be perceived as a topic in its own right, it is also addressed and discussed in this syllabus.

- Critical Success Factors; not making the mistakes many made before, a list of critical success factors is provided based on practical experiences by the authors and reviewers. In addition the test process improvement manifesto [2] is presented as a way to understand some of the critical success factors.
- Adapting to different life cycle models; test improvement approaches in agile and iterative environments are discussed and presented together with examples of where test process improvement models need to be adapted to be suitable for agile and/or iterative lifecycles

Workplace exercises

As is to be expected in a course at this level, much focus is on practical exercises. In an EL course, which for this syllabus will typically have a duration of six days, over 60% of the time will be spent on doing exercises and having some discussion sessions. A very interesting concept which will be applied to ensure participants acquire real-life practical skills and have not just experienced the non-real-life course exercises, is workplace exercises. Participants are expected to carry out exercises in their organization and report back on these in the course. There will be communication between the training provider and the participant for answering questions and checking on progress. This of course also means that the course will typically be run over a longer period of time. Examples of workplace exercises include:

- assess a test organization using either the TPI Next or TMMi model
- plan and perform assessment interviews
- create and present a summary of the conclusions (based on an analysis of the findings) and findings from an assessment
- recommend test process improvement actions on the basis of assessment results and the analysis performed
- create a test improvement plan considering change management issues, with appropriate steps and actions
- assess the critical success factors for a test improvement project.

I personally believe the ISTQB Expert level will provide added value to the testing community. Local heroes that are interested in test process improvement will in many ways benefit from this newly defined syllabus. It is about practice, demonstrating practical skills and not just theory. I hope, this short paper / column provided information to support my personal opinion. The first EL syllabus is available, this is not the end, this is just the beginning. The testing community is taking another step towards maturity.

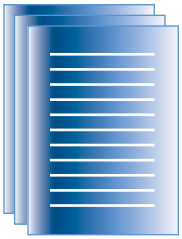
[1] Expert Level Modules Overview, V1.0 (2011), International Software Testing Qualifications Board

[2] ISTQB Certified Tester Expert Level – Certification Extension Policy, V1.0 (2008), International Software Testing Qualifications Board

[3] E. van Veenendaal, Test Improvement Manifesto, in: Testing Experience, Issue 04/08, December 2008

> biography

Erik van Veenendaal (www.erikvanveenendaal) nl) is a leading international consultant and trainer, and a widely recognized expert in the area of software testing and quality management with over 20 years of practical testing experiences. He is the founder of Improve Quality Services BV (www.improveqs.nl). 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 various domains for more than 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, vice-president of the International Software Testing Qualifications Board (2005–2009) and currently vice chair of the TMMi Foundation.



Test Configuration Formats

by Rahul Verma

This is the third installment in the Notes on Test Automation series. So far I have published one article in this series about Test Encapsulation, which focuses on making tests the most powerful part of a framework, and one about Distributed Testing Models, in which various types of implementation of distributed testing and their pros/cons were discussed (see Q3'2010 and Q1'2011 editions.)

In this article, I am going to discuss formats of test configuration as applicable to the design of Test Automation Frameworks (hereafter referred to as TAFs).

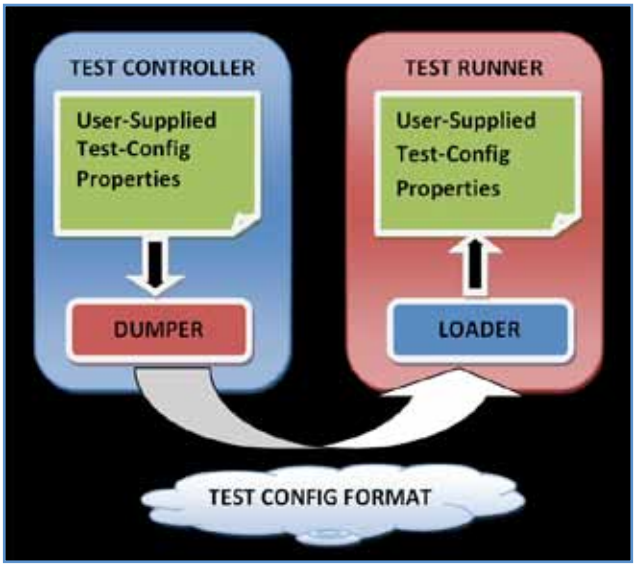
A TAF under most scenarios would provide a mechanism for the end user to configure tests for a test cycle, in which various properties can be configured. Even in basic TAFs, this is kept outside of the code for ease of use, maintainability etc.

Some common configurable properties which a TAF can provide via a test configuration include specifying the tests to be executed, the build under test, type of build, minimum priority of tests to be run, whether only tests that have known associated bugs are to be executed, and so on.

1. Design Considerations in Test Configuration Formats

A very easy way to think about configuration properties is to think of configuration as a dictionary, where the configuration property name is the key with which you can retrieve the value configured by the user (of the TAF).

The challenge involved is that under most circumstances the component of the TAF, which enables this configuration via a GUI or command line or direct editing, is different from the component which is going to consume this configuration and run the tests. More often than not, the former resides on a Master/Controller machine and the latter resides on various Test Runner machines, in case of distributed testing. For the sake of simplicity, let's give these entities names for the rest of the article:



- **Test Configuration Properties**: Properties available for direct use represented by variables/data structures in the programming language of the TAF
- **Test Configuration Format**: The format of the test configuration for storage / transmission purposes between Loader and Dumper (see below)
- **Dumper**: Part of the TAF which converts the test configuration properties into test configuration format
- **Loader**: Part of the TAF which loads the test configuration format dumped by the dumper and converts it into test configuration properties.

The diagram illustrates this:

While implementing test configuration facility, some key design questions have to be answered:

- How can we represent the user-supplied configuration properties as text information so that it can easily be converted back to the same later?
- What would be the impact of change in existing property/addition of a new property on the TAF in terms of code?
- Are we dealing with a complex configuration where the properties have hierarchical relationships?
- Do we want to support the Test Controller and Test Runner logic being written in different programming languages?
- Can the Test Controller and Test Runner exist on different operating system platforms that deal with file sharing differently?
- Do we want the test configuration format to be human readable?
- Are we going to provide a tool to create and read configuration files?

In this article, we are going to focus on the following test configuration formats that can be employed and their pros/cons:

- Text Format (with custom delimiters)
- XML
- Serialized Objects

2. Text Format

This is probably the most common format used in custom in-house TAFs. The Dumper converts the test configuration properties into a plain text format which contains data in the form of key-value(s) pairs. Typically each line represents a unique test configuration property. It usually follows the following format:

<Test Configuration Property><delimiter1><Value1><delimiter2><Value2><delimiter2>..<Valuen>

Following is an example for a text based configuration:

```
# test.cfg
BUILD_UNDER_TEST::1234
TEST_GROUPS::GP1,GP2,GP3
```

In the above example:

- BUILD_UNDER_TEST and TEST_GROUPS are test configuration properties represented as text.
- "::" is the delimiter that separates the property name from the property value
- "," is the second delimiter which separates multiple values for a single property.
- On the Test Runner machine, the Loader logic is going to read the contents of this string, by splitting based on a new line (or reading the file line by line if you share this configuration as a file rather than sending it on the network pipe), split each line based on the delimiter "::" to get the key and then split based on "," to get the value(s) for each key. This would build the test configuration properties structure back.

  A useful tip is to provide a way to add comments to a text configuration format. This helps in putting descriptive comments (# test.cfg as seen in above example). It also helps to selectively comment one or more configuration properties.

Pros/Cons

- It is very simple to relate to, does not require specialized editors, is human readable and lends itself to direct manual editing.
- It also makes the key-value relationship very obvious between testing configuration properties and their values.
- Test Controller and Test Runners can be developed in different programming languages and can reside on any sort of platform.
- Defining hierarchical relationships is very tough. For example, if you want to have different configuration settings for different kinds of tests, the Dumper might be simple to write but the Loader part is tricky and would be brittle. TAFs which use text format for such complex relationships employ the concept of sections, wherein each row is not independent on the other; rather they exist as groups of related rows. One would have to write fairly complex code to make this logic generic enough to allow for sections in the file in any order and the rows in a given section in any order. The next level of complexity would arise when all sections are now of the same type, which makes a strong case against the use of text formats altogether.
- None of the delimiters should be a part of the values. Loader logic usually implements a "Split" method which splits a string into parts based on a delimiter provided. So, in the above example, if somehow, "::" or "," is a part of the values supplied, then this would break the Loader logic.
- If you add more properties, change existing names, change the way values are expected, or change the number of values expected from 1 to many, then you would have to change both Dumper as well as Loader logic in addition to the components where test configuration properties are being populated by user and consumed by Test Runner machines. This means almost every part of the framework changes for even a small change in configuration.
- Care must be taken that extraneous spaces are stripped off, also in extraneous lines present at beginning, in the middle and at the end of the file.
- If a TAF puts this test configuration on a shared drive between Controller and Test Runner machines, it is a brittle mechanism and yet, ironically, the most common way used as observed by me. It should be sent on the network pipe created between Controller and Test Runner machines. This would prove to be very useful, especially if your test environment has different operating system platforms like Windows, Linux, Unix, Mac etc.
- If you plan to use text based configuration, try using INI / Configuration Parser modules, if available in your programming language. You could also try writing one general purpose module for the purpose which is not hardcoded for any given set of property strings/values. Your TAF could then use the module as per its custom needs.

3. XML Format

This is seen in general purpose and mature open source/free testing frameworks. The Dumper converts the test configuration properties into an XML format which contains data in the form of key-value(s) pairs represented as tags.

This is an example of an XML based configuration (with stripped xml tags for brevity):

In the example:

- Buildinfo tag is used to provide build information. It can have any number of child elements wherein each element represents a build related test configuration property.
- Testgroups tag is used to provide information about test groups (added with "tgp" tag) to be executed. Unlike the example in text format, this configuration demonstrates how you can describe properties that have different values for different test groups (LOG_PERF_STATS in this case).
- On the Test Runner machine, the Loader logic is going to feed the complete XML to an XML parser and then using the API of the XML parsing module to build the test configuration properties structure back.

```
<config>
        <buildinfo>
                <property name="BUILD_UNDER_TEST">1234</property>
        </buildinfo>
        <testgroups>
                <tgp name="GP1">
                        <property name="LOG_PERF_STATS">No</property>
                </tgp>
                <tgp name="GP2">
                        <property name="LOG_PERF_STATS">Yes</property>
                </tgp>
        </testgroups>
</config>
```

Pros/Cons

- Although XML is not meant for humans, it is human readable and for simpler formats can be edited directly (although I would suggest you evaluate whether you actually need XML for such a situation).
- Keeping the above point in mind and my earlier observation with respect to hierarchical relationships, plan to provide an easy interface to the user to provide inputs which are later converted to XML format rather than direct XML editing. Tree views are a very good way to represent XML configuration files in GUIs.
- Test Controller and Test Runners can be developed in different programming languages and can reside on any sort of platform.
- Defining hierarchical relationships is very easy unlike text based formats. Grouping of entities is also very easy as demonstrated in the example.
- Watch out for "<" and ">" in data as they are implicitly used as delimiters by the XML parsers.
- XML parsers are available in all programming languages. So, you need not write a custom parser.
- If you add more properties, change existing names or change the way values are expected, or change the number of values expected from 1 to many, then you would have to change both Dumper as well as Loader logic in addition to the components where test configuration properties are being populated by user and consumed by Test Runner machines. This means almost every part of the framework changes for even a small change in configuration. This can be partially addressed by keeping the number of tags limited. For example, instead of having "BUILD_UNDER_TEST" as a tag name, you can use it as a value of "name" attribute of "property" tag. This would enable you to introduce properties with other names and without much change in Dumper and Loader logic.
- As is the case for text formats, you should send the XML on the network pipe rather than via a file sharing mechanism. While you do this, also write XML to a file for debug and reference purpose later.

4. Serialized Object Format

I have not observed any testing framework employing this so far; of course, this might be because I haven't seen and analyzed them all. For a recent testing framework that I designed and implemented in Python, I resorted to using object serialization for test configuration and it proved to be a very useful attempt.

This is typical for object-oriented languages, although you can find modules written in traditional languages as well. You could also write a custom module of your own.

In this approach, you can represent your test configuration properties as a single data structure/object, which the Dumper would then dump as a serialized object at the Controller end of the pipe. The Loader on the TestRunner loads this and the exact test configuration properties/object becomes available. The conversion stuff is taken care of by using the object serialization modules.

A simple example is to represent the test configuration properties as a dictionary type object (in traditional programming languages like Perl, it is called a hash). Populate the key-value pairs in this dictionary based on user-supplied input. Now, you can directly dump this as a serialized object using a language's corresponding module. Similarly at the Test Runner end, you would load the object using the same module.

Pros/Cons

- Serialized objects are not human readable. In some languages like Python, serialized objects are semi-human-readable. Whatever be the case, they are not meant to be manually editable as editing might break the format.
- Keeping the above point in mind, you must provide an easy interface to the user to provide inputs which are later converted to the serialized object format. Also for debugging purposes, you should also provide an interface to load a serialized object to see what was configured.
- Test Controller and Test Runners would need to be developed in the same language because the core format of serialized objects would most likely be different in different languages. To avoid this, you might have to write custom modules in both languages which use the same serialized format, which in my opinion would be overkill (use XML instead), unless you want to develop some proprietary stuff for commercial purposes.
- Defining hierarchical relationships is very easy, unlike text based formats, with the help of nested data structures.
- There are no problems related to delimiters at all in the Loader. The Loader in this case would most likely be a time-tested piece of code, as it is a part of the core language platform rather than developed by a hobbyist-tester-turned-coder like me.
- Addition of properties would not have any impact on Loader or Dumper logic at all. So, you can make any changes without much rework.

5. What's Next?

Next, I will write on how to run tests written in any language on top of a framework written in a given language using files and IPC.

> biography

Rahul Verma
is a Senior QA Technical Lead with the Anti-Malware Core team of McAfee Labs, India. He runs the Testing Perspective (www.testingperspective.com) website for which he was awarded the Testing Thought Leadership Award by PureConferences. He has special interest in performance testing, security testing, Python and design of test automation frameworks. Rahul has presented at several conferences, organizations and academic institutions including GTAC, CONQUEST, STeP-IN, ISQT, TEST2008, Yahoo! India, McAfee and IIT Madras. He is a member of the Working Party for the ISTQB's Advanced Level Test Certifications and is the President of Indian Testing Board's Bangalore TMM Chapter.
rahul_verma@testingperspective.com

Agile TESTING DAYS

November 14–17, 2011
Potsdam (near Berlin), Germany

www.agiletestingdays.com

Agile TESTING DAYS

November 14–17, 2011
in Potsdam (near Berlin), Germany

The Agile Testing Days is an annual European conference for and by international professionals involved in the agile world. This year's central theme is **"Interactive Contribution"**.

Please visit our website for the current program.

Registration is open!
Catch the Early Bird fee and register by August 31, 2011!

Agile Testing Days 2011 –
A Díaz & Hilterscheid Conference

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

info@agiletestingdays.com
www.agiletestingdays.com

Tutorials – November 14, 2011

"Hooray! We're Agile Testers! What's Next? Advanced Topics in Agile Testing"
Lisa Crispin

"Making Geographically Distributed Projects Work"
Johanna Rothman

"Influence Strategies for Practitioners & "Patterns for Improved Customer Interaction"
Linda Rising

"Winning big with Specification by Example: Lessons learned from 50 successful projects"
Gojko Adzic

"Acceptance Testing: From Brains to Paper and Paper to Computer"
Lasse Koskela

"Transitioning to Agile Testing"
Janet Gregory

"Dealing with Differences: From Conflict to Complementary Action"
Esther Derby

"Critical Thinking Skills for Testers"
Michael Bolton

"Introduction to BDD"
Elizabeth Keogh

"Agile Management: Leading Software Professionals"
Jurgen Appelo

Conference (Day 1) – November 15, 2011

Time	Track 1	Track 2	Track 3	Track 4	Vendor Track
08:00–09:25	Registration				
09:25–09:30	Opening				
09:30–10:30	**Keynote: "Agile Testing and Test Management"** – Johanna Rothman				
10:30–11:30	**"What Testers and Developers Can Learn From Each Other"** David Evans	**"Specification by Example using GUI tests – how could that work?"** Geoff Bache & Emily Bache	**"Agile Performance Testing"** Alexander Podelko	**"SQL PL Mock – A Mock Framework for SQL PL"** Keith McDonald & Scott Walkty	**"The roles of an agile Tester"** Sergej Lassahn (T-Systems Multimedia Solutions GmbH)
11:30–11:50	Break				
11:50–12:50	**"Do agile teams have wider awareness fields?"** Rob Lambert	**"Experiences with Semi-scripted Exploratory Testing"** Simon Morley	**"Design For Testablity is a Fraud"** Lior Friedman	**"Using agile tools for system-level regression testing in agile projects"** Silvio Glöckner	Talk 5.2
12:50–14:20	Lunch				
14:20–15:20	**Keynote: "Who do You Trust? Beware of Your Brain"** – Linda Rising				
15:20–16:20	**"Top Testing Challenges We Face Today" "** Lloyd Roden	**"Session Based Testing to Meet Agile Deadlines"** Mason Womack	**"Unit testing asynchronous JavaScript code"** Damjan Vujnovic	**"Automated Functional Testing with Jubula: Introduction, Questions and Answers"** Alexandra Imrie	Talk 5.3
16:20–16:40	Break				
16:40–17:40	**"I don't want to be called QA any more! – Agile Quality Assistance"** Markus Gaertner	**"TDD with Mock Objects: Design Principles and Emerging Properties"** Luca Minudel	**"Agile on huge banking mainframe legacy systems. Is it possible?"** Christian Bendix & Kjær Hanse	**"Automated testing of complex service oriented architectures"** Alexander Grosse	Talk 5.4
17:40–18:40	**Keynote: "Appendix A: Lessons Learned since *Agile Testing* Was Published"** – Lisa Crispin & Janet Gregory				
18:40–18:45	Closing Session				

The Agile Testing Days 2011 "MIATPP"- Award

Please vote for the "Most Influential Agile Testing Professional Person" in 2011. At the Agile Testing Days in November 2011 we will award this trophy for the first time. **VOTE NOW!**

Who do you think is the most formative agile person in your testing community? Send the person's name including a short explanation to **award@agiletesting-days.com** until September 30th and you could be the winner of 2 free tickets for the Agile Testing Days 2011!

Exhibitors & Supporters 2011

T-Systems · testing experience · Agile Record · Díaz Hilterscheid

Want to exhibit?

If you'd like to be an exhibitor at Agile Testing Days 2011, please fill in the form which you can find in our exhibitor brochure and fax it to +49 (0)30 74 76 28-99 or e-mail it to **info@agiletestingdays.com**.

» Download our exhibitor brochure at **www.agiletestingdays.com**

Agile TESTING DAYS

Conference (Day 2) – November 16, 2011

Time	Track 1	Track 2	Track 3	Track 4	Vendor Track
08:00–09:25	Registration				
09:25–09:30	Opening				
09:30–10:30	**Keynote: "People and Patterns"** – Esther Derby				
10:30–11:30	**"About testers and garbage men"** Stefaan Luckermans	**"ATDD and SCRUM Integration from a traditional Project methodology"** Raquel Jimenez-Garrido	**"Test automation beyond GUI testing"** H. Schwier & P. Jacobs	**"Micro-Benchmark Framework: An advanced solution for Continuous Performance Testing"** Sven Breyvogel & Eric Windisch	Talk 5.5
11:30–11:50	Break				
11:50–12:50	**"Do we just Manage or do we Lead?"** Stevan Zivanovic	**"Agile ATDD Dojo"** Aki Salmi	**"Make your automated regression tests scalable, maintainable, and fun by using the right abstractions"** Alexander Tarnowski	**"Beyond Page Objects – Building a robust framework to automate testing of a multi-client, multi-lingual web site"** Mike Scott	Talk 5.6
12:50–14:20	Lunch				
14:20–15:20	**Keynote: "Haiku, Hypnosis and Discovery: How the mind makes models"** – Elizabeth Keogh				
15:20–16:20	**"A Balanced Test Strategy Strengthens the Team"** Anko Tijman	**"Effective Agile Test Management"** Fran O'Hara	**"Sustainable quality insurance: how automated integration tests have saved our quality insurance team."** Gabriel Le Van	**"Automate Testing Web of Services"** Thomas Sundberg	Talk 5.7
16:20–16:40	Break				
16:40–17:40	**"Testing your Organization"** Andreas Schliep	**"Get your agile test process in control!"** Cecile Davis	**"Measuring Technical Debt Using Load Testing in an Agile Environment"** Peter Varhol	**"Real loadtesting: WebDriver + Grinder"** Vegard Hartmann & Øyvind Kvangardsnes	Talk 5.8
17:40–18:40	**Keynote: "Five key challenges for agile testers tomorrow"** – Gojko Adzic				
19:00–23:00	Chill Out/Award Event				

Collaboration Day – November 17, 2011

Time	Track 1	Track 2	Track 3	Track 4
08:00–09:25	Registration			
09:25–09:30	Opening			
09:30–10:30	**Keynote: "No More Fooling Around: Skills and Dynamics of Exploratory Testing"** – Michael Bolton			
10:30–11:30	**Open Space** – Brett L. Schuchert	**Testing Dojos** – Markus Gaertner, Alex Bepple and Stefan Roock	**Coding Dojos** – M. Gaertner, A. Bepple and Stefan Roock	**TestLab** – B. Knaack & J. Lyndsay
11:30–11:50	Break			
11:50–12:50	**Open Space** – Brett L. Schuchert	**Testing Dojos** – Markus Gaertner, Alex Bepple and Stefan Roock	**Coding Dojos** – M. Gaertner, A. Bepple and Stefan Roock	**TestLab** – B. Knaack & J. Lyndsay
12:50–13:50	Lunch			
13:50–14:50	**Keynote: "Stepping Outside"** – Lasse Koskela			
14:50–16:50	**Open Space** – Brett L. Schuchert	**Testing Dojos** – Markus Gaertner, Alex Bepple and Stefan Roock	**Coding Dojos** – M. Gaertner, A. Bepple and Stefan Roock	**TestLab** – B. Knaack & J. Lyndsay
16:50–17:50	**Keynote: "The 7 Duties of Great Software Professionals"** – Jurgen Appelo			
17:50–18:00	Closing Session			

Agile TESTING DAYS

November 14-17, 2011
Potsdam, Germany
www.agiletestingdays.com

Save the date, discover the place.

Foto: HPI / K. Herschelmann

Potsdam

The most beautiful place for bright minds

Mr. Net is a Potsdamer! This interactive sculpture, created by the Spanish artist Jaume Plensa, is located on the grounds of the Hasso-Plattner-Institute, an unmatched centre of academic excellence for IT systems-engineering in Germany. The combination of first class alumni, innovative business start-ups, and successful companies makes Potsdam a premium IT location. Come to Potsdam and discover the city for yourself: as a lovely holiday destination, as a multi-faceted event location, or as an attractive place for your business.

Potsdam is a unique mixture of world heritage and high-tech, of science and business, of Brandenburgish cultural milieu and Mediterranean charm. It is an attractive city and a premium location for business with outstanding prospects for future growth. Potsdam is also a growing city with creative potential. Located right next door to the metropolis Berlin, Potsdam offers everything you need to succeed. Potsdam: a great place for great ideas.

This great city welcomes you warmly!

POTSDAM

State Capital Potsdam
Department of Business Development
economy@potsdam.de I www.potsdam.de

Accepting and embracing change in automated acceptance tests with Jubula

by Alexandra Imrie

Automated GUI testing has a terrible reputation. If you mention it at a table of testers or developers, then you'll almost certainly hear that "everything breaks every time you make a change" and that "tests are impossible to maintain". Not only does it have this awful reputation, but it is also often misrepresented. All too often the perception of GUI testing is to *test the GUI* – is the button there, is the text field red etc.

GUI tests are not superficial tests

The misrepresentation is usually the easiest to clear up. If we think less in terms of testing the GUI and more of testing *through* the GUI, then the importance of GUI testing becomes clear. Our customers and users will only be able to work with the software using the GUI. Yes, it's important for them that interactions at the unit level are correct, but their scope of reference is the interface they have in front of them. The workflows and use cases requested in the specifications must be executable using the interface, and to be sure of this, we have to test them.

I heard an excellent metaphor for GUI testing. If we think of the application as a pond, then the unit level is close to the bottom of the pond. We can get very close to the intricate details, but are so far away from the surface that we can easily lose the big picture. Just testing the existence or properties of components in the GUI itself is the equivalent of skimming a stone across the surface of the pond. An acceptance test through the GUI, on the other hand, can be thought of as taking a large stick and poking all the way down to the depths of the pond. Starting from the user perspective, we go right to the bottom, passing every layer of business logic on the way. If there's a problem, then we can send a diver to take a closer look.

Change is normal

The terrible reputation may take some more work to clear up. The two main quibbles noted above are both related to changes. The first complaint is that changes in the code require changes to the tests. If we look at this objectively, we find this is a completely normal and desired part of the development process. It is natural that a change to a requirement may require a change in the code. In the same way, a change to requirements or code may also require a change to the tests. No test at any level can be written just once and yet remain up-to-date despite changes. If we accept that our tests need to be updated to keep in line with continued development, the only question is how much effort that change requires.

A question of effort

Minimizing the effort required to update systems isn't a unique concern of testing though. If we look at the best practices we know from software development, many of them are in place to ensure that any maintenance work is easy to do. Just some of these best practices include:

- naming items to make them easily understandable
- combining instructions to make logical structures
- parametrizing information within structures to make them more generic
- referencing structures instead of rewriting or copying them

If we stick to these best practices, then when it comes to changing something we know that:

- we can search for it and understand it based on its name
- we can add, change and remove instructions or parameters from logical structures and have these changes propagated to all places where the structure has been referenced
- existing logical structures may provide the basis for newly required logical structures

These best practices for software development are just as relevant and usable for automated testing. If tests are easy to maintain, then they can continue to run despite changes to the software. Continuously running tests deliver constant information which feeds back into the process to inform us of any questions, issues or errors from that all-important customer perspective (Figure: quality_curve.jpg).

Where the best practices get forgotten

One of the reasons for the bad reputation of automated GUI testing is almost certainly the predominance of the capture-replay method. While the aim to let the whole team participate in test automation is laudable, a recorded script is by no means an auto-

mated test. It is full of repeated actions, redundancies, mistakes, hard-coded data and implicit assumptions. Although the actions are recorded, the intentions and intelligence aren't. In short, none of the best practices we strive to uphold for our software are in place. Although the script may run once or even a few times, it will soon break. As mentioned above, changes are normal – it is the effort associated with them that is critical. Looking at a recorded script, it is painfully obvious that adapting it to make it maintainable is not very different from rewriting the script from scratch.

Writing tests in program code, however, requires a skilled software developer. The focus on the user perspective is quickly lost and managing the test code can often become a project in itself.

The secret behind successful long-term GUI tests is to preserve the focus on team collaboration to ensure that the user perspective is constantly represented. At the same time, we have to get rid of the effort associated with writing and maintaining program code whilst conforming to the best practices we know from writing software projects.

Introducing Jubula

Jubula is a newly created open source project at the Eclipse Foundation for automated acceptance tests that can be written by all team members according to best practices, without any coding effort. It was created from the core parts of the commercial tool GUIdancer and forms the basis for the continued development of GUIdancer.

Instead of recording or coding, Jubula tests are written using drag and drop from a library of modules. Each module is an executable action such as selecting, clicking, checking, dragging&dropping or entering text. Modules can be renamed for readability, combined to form reusable sequences and are generic in terms of data to make them more flexible. The GUI object recognition (also a frequent pain point) uses a heuristic algorithm to locate even changed objects and is also managed centrally to minimize any necessary maintenance work.

This approach ensures the longevity of the test. Recurring sequences are stored centrally so that changes are propagated throughout the test, and the test structure itself is easy to read because of the natural language naming. The removal of programming activities means that all team members can collaborate on the test automation and also reduces the effort associated with maintaining tests.

A test automated with Jubula is a repeatable verification that a specific workflow functions correctly through the medium that the customer will also use. The intelligence of a manual tester is incorporated via various check actions, dynamic synchronization options and structures to react to planned or unplanned deviations within the workflow. It can be used as an acceptance test to (help) confirm that a feature is complete, and as a regression test to ensure that already completed features are not adversely affected by new developments.

Embracing the change

Change is inevitable, but this shouldn't be used as an excuse to ignore the perspective of those who will use the software we are developing. Continuous acceptance testing provides valuable information about the state of the software and uncovers questions and issues that are better treated earlier rather than later. If done correctly, it is a safety net that accompanies each new build or version to provide software that lets the customer work as desired.

Links

http://www.eclipse.org/jubula

> biography

Alexandra Imrie
earned a degree and an MA in linguistics from York University before starting work at Bredex GmbH, where she is a trainer and consultant for automated testing and test processes. When she is at the office, she is a part of the development team, representing the customer / user perspective for software. Alex helps to write user stories, brings feedback from the field and enjoys contributing to the emerging software in terms of testing. Alex frequently represents Bredex at conferences, where she talks about agility and testing from project experience. She is also involved in event organization and customer communication. Two of her main interests in the world of software development are how to make software user-friendly and how to bring agility to testing processes.

Business Scenario Technique

by Debbie Powell

Everyone has their own way of doing things...such as writing business/testing scenarios. There are many different techniques that can be used. It is often difficult to explain to others your technique. In the following, you will find the technique that has been most successful for several of my colleagues and me. This technique helps to ensure that the tests will cover all components of the business requirement.

1. How should the business scenarios be written?

For example, the requirement states:

Prevent Access and Updates to Locked Account - The system prevents access and displays an error message when the user attempts to open, view, or change a locked hierarchical account, large account, or subscription via <application>.

2. The easiest way to determine scenarios is to break the requirement into Functions, Responses and Conditions. The three elements should be straightforward to spot:

- Functions can be described explicitly with phrases like "the system will allow the user to add, amend and delete records...".
- Responses often come early in sentences with phrases like "the system will reject entries with ...." or late in sentences like "... entries with ... will be rejected".
- Explicit Conditions are often presented in constructs like "if ... then ...." phrases.
- Implicit elements are obviously more difficult to spot: 'if... then' phrases imply some different behavior if the condition is false.
- Some verbs like 'update' imply system functions, whilst others, such as 'reject', indicate a response with the relevant condition described elsewhere (Gerrard, 2000).

Functions
1. User attempts to open the account.
2. User attempts to view the account.
3. User attempts to change the account.
4. User attempts to open the subscription.
5. User attempts to view the subscription.
6. User attempts to change the subscription.

Responses
7. Access is prevented.
8. Error message is displayed.

Conditions
1. Hierarchical account is locked.
2. Large account is locked.
3. Subscription is locked.

3. Having analyzed a requirements section into three lists, the next task is to rationalize each list in turn. The purpose here is to remove duplicate functions and conditions and give each element a brief description which is consistent with its origin in the requirements. The process is quite simple:

- Select each function in turn
- For each condition in the list, consider whether the condition is relevant to the function selected. If it is, we add a new row to the table of behaviors.
- Scan the list of responses, and select the response which the requirements document predicts, if there is one.

Repeat the above for every function and condition building up the table of behaviors. (Gerrard, 2000)

Table of Behaviors		
Functions	**Responses**	**Conditions**
User attempts to open the account.	Access is prevented. Error message is displayed.	Hierarchical account is locked.
User attempts to view the account.	Access is prevented. Error message is displayed.	Hierarchical account is locked.
User attempts to change the account.	Access is prevented. Error message is displayed.	Hierarchical account is locked.
User attempts to open the account.	Access is prevented. Error message is displayed.	Large account is locked.
User attempts to view the account.	Access is prevented. Error message is displayed.	Large account is locked.
User attempts to change the account.	Access is prevented. Error message is displayed.	Large account is locked.
User attempts to open the subscription.	Access is prevented. Error message is displayed.	Subscription is locked.
User attempts to view the subscription.	Access is prevented. Error message is displayed.	Subscription is locked.
User attempts to change the subscription.	Access is prevented. Error message is displayed.	Subscription is locked.

(Gerrard, 2000)

4. Having organized the functions, responses and conditions into a table of behaviors, the next task is to convert them into business scenarios. Essentially, you will combine the functions and conditions to create the information for the Actions/Events/Criteria column and enter the responses in the Reactions/Expected Results/Comments column. You will then enter the reference information and application impact information.

Actions/Events/Criteria	Reactions/Expected Results/Comments	Reference	App. Impact
User attempts to open a locked hierarchical account.	Access is prevented. Error message is displayed.	<reference>	<application>
User attempts to view a locked hierarchical account.	Access is prevented. Error message is displayed.	<reference>	<application>
User attempts to change a locked hierarchical account.	Access is prevented. Error message is displayed.	<reference>	<application>
User attempts to open a locked large account.	Access is prevented. Error message is displayed.	<reference>	<application>
User attempts to view a locked large account.	Access is prevented. Error message is displayed.	<reference>	<application>
User attempts to change a locked large account.	Access is prevented. Error message is displayed.	<reference>	<application>
User attempts to open a locked subscription.	Access is prevented. Error message is displayed.	<reference>	<application>
User attempts to view a locked subscription.	Access is prevented. Error message is displayed.	<reference>	<application>
User attempts to change a locked subscription.	Access is prevented. Error message is displayed.	<reference>	<application>

Helpful Links:

The Unifying Role of Scenarios in Conceptual Design: http://www.humanfactors.com/downloads/apro42.htm

Software QA and Testing Resource Center: http://www.softwareqatest.com/

Bibliography:

Gerrard, Paul. (2000). Testing Requirements. Retrieved February 21, 2007, from http://www.testinginstitute.com/Bug_Free_Zone/journal/Requirement_Based_Testing/Test%20Requirements.pdf .

> biography

Debbie Powell
has worked as a Senior Test Analyst at Convergys Corporation since 2006 and has experience in software development and testing since 1984. She earned her Bachelor of Science in Business Information Systems from University of Phoenix, Master of Art in Computer Resources and Information Management and Master of Art in Educational Technology from Webster University. Feedback is welcome at debpowell5@gmail.com.

Improving the Test Process

by Dhiraj Kale

Testing can often be a trouble¬some and uncontrollable process. It takes time, costs a lot more than planned, and offers insufficient insight into the quality of the test process - possibly putting the quality of the information system being tested and business process itself at risk. So what is the solution to this business challenge?

Improve your test process

Many organizations realize that improving the test process is essential for ensuring the quality of information systems and overall business processes. However, in practice, it's often challenging to define the steps required to improve and control the process, and the order.

In today's business environment, develop¬ments occur at a very fast pace. The devel¬opers' productivity is continuously increasing and clients are demanding even higher lev¬els of quality. Even if your current test process is satisfactory, this process will need improve¬ment in the future.

Organizations should review their testing process and be proactive and forward thinking. They might prefer to have a defined standard for testing or a continuous improvement program that is constantly evolving to meet both their own as well as the customers' needs. Of course there's very little point in reviewing your testing process if the basics are not in place to begin with, but if you are looking for a place to start then **some** of Insight's top 10 tips are a must.

Planning Ahead - Ensure at the very least that the organization has a basic test process in place as this helps to clarify testing responsibilities, overall approach and will lead to less headaches. Although you'll have room for improvement, it's a good start.

Early Days - Getting testers involved at the earliest stage, i.e. at the requirements gathering stage or even before, has proven to be beneficial, as it's cheaper to find and correct defects at this stage of the project.

Risky Business - Consider running a Risk Workshop so that the high-risk areas are at least identified at the outset. The 'old risk versus reward' theory comes to mind here.

Focus the Mind - A test strategy will identify what's to be tested and focuses the mind on what the testers are trying to achieve.

Buy-In - Ensure that your Project Managers buy into your testing process.

Communication is Key - Don't forget to have regular meetings with all of the stakeholders so that items don't get de-railed.

Overall - We came away feeling that we can use some of the above in our own process.

The TPI® Next model

TPI® Next is the Sogeti model for the business-driven improvement of testing that offers insight into the maturity of the current test process and identifies improvement actions to accomplish the desired test maturity level and business goals [Sogeti 09]. Through extensive global adoption since 1998, TPI® Next has become the de-facto industry standard for test process improvement.

Recent updates to TPI® Next

Since the original release of TPI®, new topics have had increased impact on test best practice. Examples include (test) outsourcing, Agile testing, multiple test processes, end-to-end testing, etc. These topics are covered by the updated model, TPI® Next.

Additionally, there are descriptions of how to deal with specific test types and/or test levels, like evaluation, integration testing to provide you with practical tips and tricks for your specific situation. Enablers are a new addition to TPI® Next and clarify the relationship between testing and adjacent processes and how they can benefit from each other. This supports organizations that are already using software process maturity models such as CMMI® or ISO/IEC 15504.

The TPI® model has been developed based on the knowledge and the experience of Sogeti and its clients. The model offers insight into the "maturity" of the test processes within your organization. Based on this understanding, the model helps you to define gradual and controllable improvement steps.

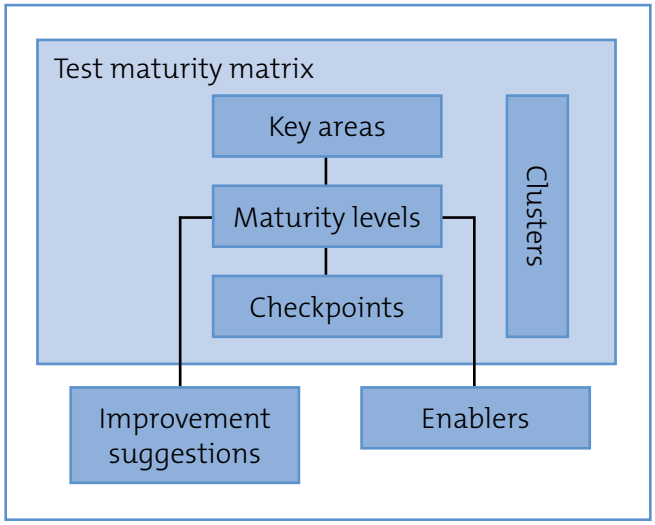
The TPI® Next model offers business-driven test process improve-

ment through the concept of clustering, which aligns the order of improvement efforts with your business drivers.

Features:

- Business-driven test process improvement
- Covers complete test process (16 key areas)
- Objective (157 pre-defined checkpoints)
- Detailed: stepwise improvement possible
- Logical order of improvements
- Quick start to improvement
- Benchmark possibility (worldwide no.1 model)

The model is depicted as follows:



TPI® Next considers the different aspects of the test process, such as the use of test tools, design techniques or reporting. By evaluating various aspects, the strengths and weaknesses of the test process become clear.

These aspects are called the key areas. The TPI® Next model has 16 key areas. These key areas are as follows:

1. Stakeholder commitment
2. Degree of involvement
3. Test strategy
4. Test organization
5. Communication
6. Reporting
7. Test process management
8. Estimating and planning
9. Metrics
10. Defect management
11. Testware management
12. Methodology practice
13. Tester professionalism
14. Test case design
15. 15. Test tools
16. Test environment

Test Maturity Matrix

The TPI® Next model uses a test maturity matrix to provide a visual overview of the key test areas and their respective maturities. The maturity matrix lists the 16 key TPI® Next test process areas. It then shows the overall maturity level based on the 16 key areas, i.e. four maturity levels - Initial, Controlled, Efficient and Optimizing.

The combination of key areas, maturity levels and checkpoints are used to identify the strengths and weaknesses of the current test process, and to help define actions for improvement.

Key Area	Initial	Controlled	Efficient	Optimizing
1. Stakeholder commitment				
2. Degree of involvement				
3. Test Strategy				
4. Test organization				
5. Communication				
6. Reporting				
7. Test process management				
8. Estimating and planning				
9. Metrics				
10. Defect Management				
11. Testware management				
12. Methodology practice				
13. Tester professionalism				
14. Test case design				
15. Test tools				
16. Test environment				

Testing

is Now Radically

Easier

Telerik Test Studio

Easy Test Creation

- Test web & desktop apps - HTML | AJAX | Silverlight | WPF
- Intuitive point-and-click recording
- Record once, run against multiple browsers

Easy Test Maintenance and Reporting

- Element abstraction and reuse
- Remote scheduling, execution and results reporting
- Seamless QA-Developer collaboration

Download your free trial and get 20% off Test Studio at:
www.telerik.com/TestingExperience

telerik
deliver more than expected

Filling in the matrix makes it easier to evaluate the improvement proposals. The requirements for each level are defined in the form of checkpoints which are questions that need to be answered positively in order to qualify for that level. These checkpoints make an objective classification by maturity level possible.

Improvement actions can be defined in terms of desired higher levels of test process maturity. Checkpoints help determine the actions required to achieve a higher level. Additionally, the model offers improvement suggestions to support test process improvement, which include a list of tips and ideas that can help in the effort to achieve the desired level.

TPI® Next Assessment Approach

The following outlines Sogeti's generic approach for a typical TPI Assessment:

- **Obtain awareness for improvement** - Agree on the need for improvement and get commitment from all involved that improvement is necessary
- **Determine the scope for improvement** - Establish the scope for the assessment, identifying the areas to be targeted for improvement
- **Interview all your personnel** - Interview all your key personnel that have direct or indirect influence over your test processes
- **Gather documentation** - Gather and collate the key documents which feature in your test processes:
  - Test Strategies
  - Test Plans
  - Test Scripts and other influencing documents where relevant.
- **Establish your current test maturity** - Using the TPI® Next model and the Test Maturity Matrix, chart your organization's current test maturity, highlighting areas of strength and weakness
- **Define improvement actions** - Using the TPI® Next model and the Test Maturity Matrix, define the improvement actions that will get your organization's test maturity to the desired level
- **Present assessment findings** - The key output of this assessment will be a detailed TPI Assessment report which will include:
  - Your organization's current test maturity
  - Your organization's desired test maturity
  - Action plan outlining how to get from your current to your desired level of test maturity

TPI® helps organizations to realize the desired test maturity levels and achieve more streamlined and efficient processes. It enables organizations to optimize their testing performance and identifies the strong and weak points of the current situation in an organization, while suggesting the necessary improvements. The improvements are then implemented according to a plan and are monitored to meet the business goals.

With the ever changing business eco-system and dynamic needs of customers, business goals of an organization might change, and TPI addresses these by suggesting improvements to the test processes.

Benefits of Test Process Improvement

- Increases effectiveness and efficiency of test activities
- Aligns testing with organizational priorities and with other project processes
- Improves both real and perceived value of testing to the organization
- Improved overall software quality
- Reduced down time
- Reduced errors
- Lower production maintenance overheads
- Enhanced compliance
- More efficient and effective business operations
- Long-term reduction in the cost of testing
- Long-term reduction in the length of the test process
- Increased efficiency with respect to the development and testing life cycle and supporting processes
- Increased quality of code into production and code into each of the test phases
- Effective base-lining and evaluation tool to determine if the test improvements and benefits envisaged are being achieved.

Reference:

- [Sogeti09] Sogeti, "TPI Next – Business Driven Test Process Improvement", UTN Publishing, 2009, ISBN 90-72194-97-7

Trademarks:

> biography

Dhiraj Kale

I am currently working as a Senior Technical Associate at Techmahindra Pune. I have a bachelor degree in electronics and telecommunication and have worked as a senior tester with more than 5 years of experience in the fields of test automation, manual testing and tool support. I have gathered knowledge of the investment banking, capital market and telecom domains.

The Need for an Abstraction Layer Between Requirements and Tests

by Jeffrey H. Lucas

Business and functional requirements serve an important role in the software development life-cycle. Methods for creating unambiguous, testable requirements have been well documented [1] [2]. They provide a forward-looking basis for the development of software designs. The requirements are normally created without reference to the implementation approach. This gives the developer the greatest amount of flexibility in providing innovative ways of satisfying the needs of the stakeholders. The requirements may not be as well written in many development efforts. This can manifest itself in overly complex requirements (which leads to the need for partial requirement tracking), ambiguous requirements, or missing requirements.

Even tracing test cases to well-written functional requirements can introduce problems. All development efforts involve multiple deliveries to the customer (e.g. prototypes, beta releases, and cyclical maintenance releases). Interactions with these early releases create expectations among stakeholders that are not tracked in the existing requirement management tools. Tracking changes in the implementation of earlier releases is important, not only from customer expectations, but also in targeting behavior that can have an effect on training and user help documentation.

Improving the Association between Requirements and Test Cases

One way to effectively test application functions that do not have appropriate requirement traceability is through a layer of objects between requirements and tests. These objects incorporate a combination of both function points from requirements and interface points from the implementation. In this article, they will be referred to as "touch points", but equivalent implementations may have many names in different organizations.

The concept of adding an abstraction layer between tests and requirements has been introduced before. For example, Erik Runhaar talked about using "focus points" as a means to more fully describe requirement aspects that are identified as important to the customer [3]. In addition, some test tools include screen-shot capture functions that can be used in place of requirement traceability when requirements are missing [4]. This abstraction layer should be a formal part of software testing that includes training testers in the proper usage of such techniques as a part of the curricula and incorporating the layer into standard test tool suites.

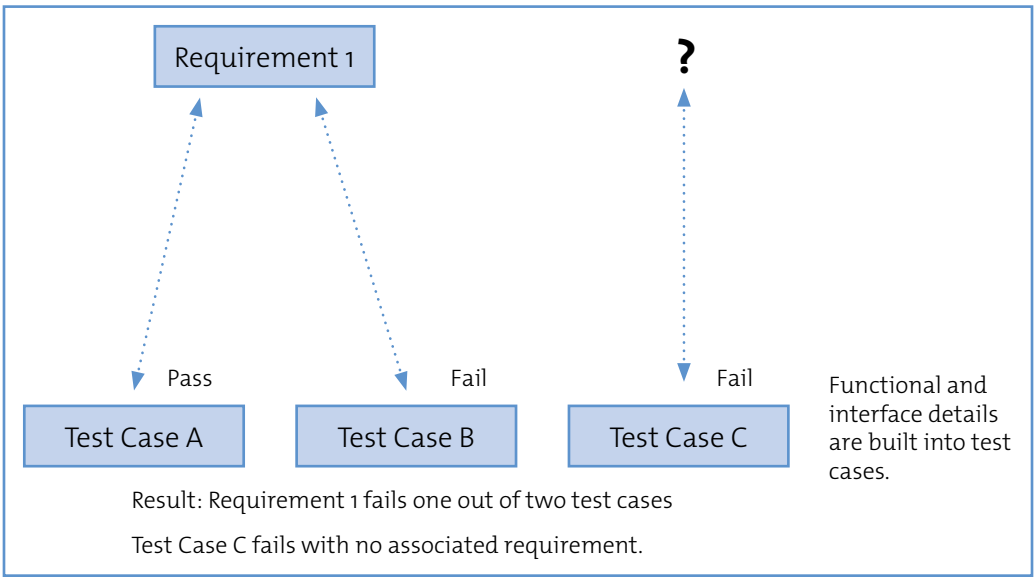


Figure 1: Direct Association Between Requirement and Tests

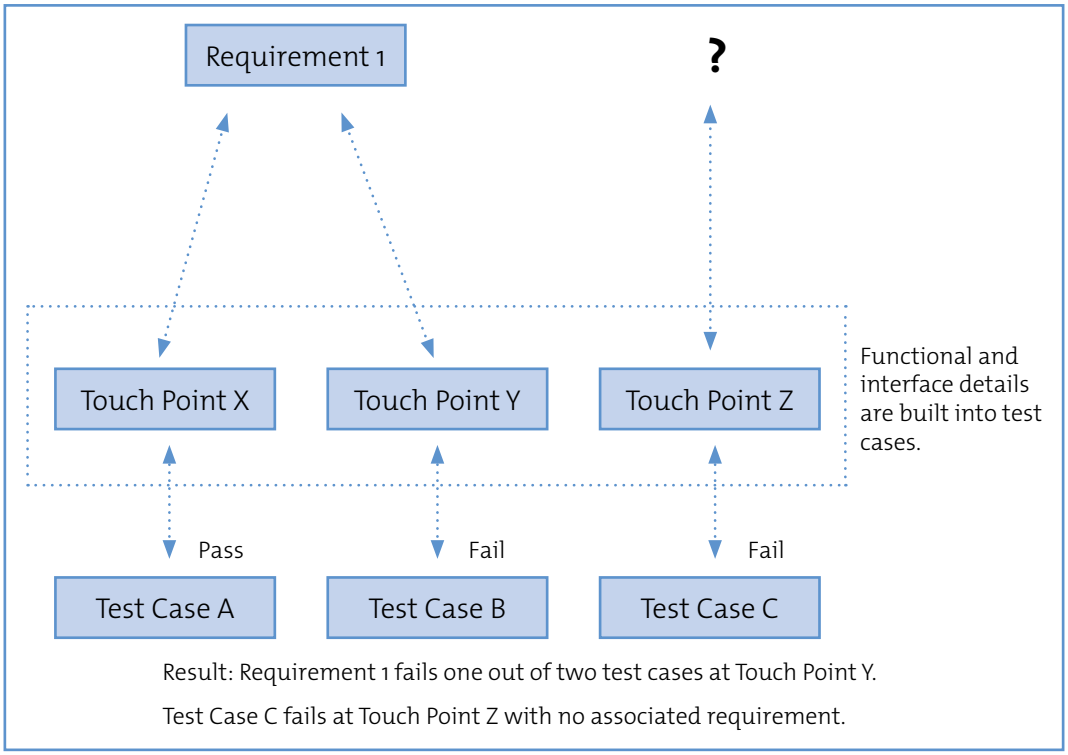


Figure 2: Intermediate Association to Touch Points

The Abstraction Layer in Practice

Touch points are structured *from the user's perspective*. Hierarchical categories are used to define the interface details for that function. Functional details are specified at the leaf# category or in a separate comments area. This format normally provides sufficient information to perform testing using exploratory or checklist test techniques without the creation of formal test procedures. More complex tests that do require formal procedures or automated test scripts can be linked to the touch points with a hyperlink to the test case. This touch point specification can be used to more easily track and communicate the location of defects or planned software changes that may impact multiple areas of the application interface.

Sometimes an area of software functionality exists that has no direct relationship to formal requirements. For example, Rex Black warned that the Agile software development approach (which emphasizes face-to-face meetings over detailed requirements development) could introduce just such requirement traceability issues if the test oracle is not well documented [5]. The touch point layer envisioned here is not intended to be a replacement for requirements – it is important that testers keep their testing rooted in the customer needs and not on the implementation [6]. Instead, the touch point layer is intended to provide a means to quickly document changes in the behavior of the application as a part of the test basis. This will help identify deviations from documented requirements or the implementation from previous deliveries.

Basing the touch points on both function points and interface points provides a solution to the additional problem of partial requirement tracking. Once the association between requirements and touch points is validated, association of touch points to test cases provides independent verification of each portion of the requirement (as illustrated in Figure 1 and Figure 2). This has the potential to make the development of test cases and the training of new testers simpler, faster, and less costly.

Integration of Automated Testing with Touch Points

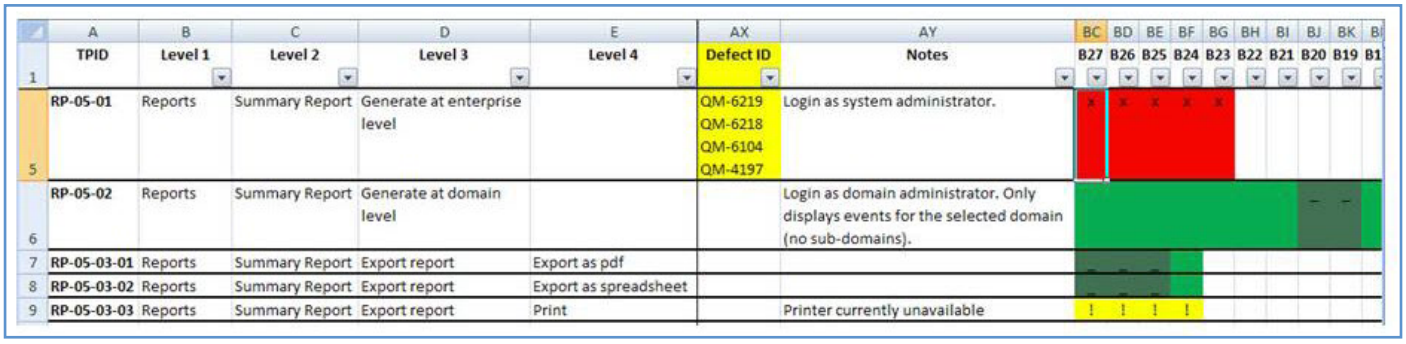
The development of effective regression test strategies is crucial for iterative deployments, especially software maintenance releases [7] [8]. This is where software test automation can provide significant benefits to the overall test effectiveness. However, the automated script developer must understand the limitations of such an approach in verifying requirements.

The concept of "fully" testing any application function purely through automated means is difficult to document and verify, especially in the area of Graphical User Interface (GUI) test automation. This is due to the fact that test automation can be equated to a near-sighted sprinter – it goes very fast but only sees what is directly in front of it. Issues that would be obviously wrong to a human tester will not be "seen" by the automation unless programmed to specifically detect those occurrences.

Instead, automation creates slices through the test domain space (which is defined here as a combination of the function points and interface points). Each of these slices may "touch" a requirement, but not fully test it. Cause-Effect Graphs and Decision Tables are powerful techniques for designing the minimum number of test cases for full coverage of interrelated, complex requirements [9]. The use of touch points with these techniques could help in documenting the parsing of complex requirements for partial testing and in identifying alternative interface paths.

Suggested Future Test Tool Enhancements

Implementing this touch point abstraction layer manually in a spreadsheet format is only practical for a small product development team. Maintaining touch points manually is labor intensive and requires a good deal of organization. As a result, the best approach for requirement traceability using this method is at a relatively high level with detailed points specified only for very complex functions. When combined with risk assessment techniques, that approach is consistent with general practice for effective requirement tracking except for those applications that are safety



	A	B	C	D	E	AX	AY	BC	BD	BE	BF	BG	BH	BI	BJ	BK	BL
1	TPID	Level 1	Level 2	Level 3	Level 4	Defect ID	Notes	B27	B26	B25	B24	B23	B22	B21	B20	B19	B1
5	RP-05-01	Reports	Summary Report	Generate at enterprise level		QM-6219 QM-6218 QM-6104 QM-4197	Login as system administrator.	x	x	x	x	x					
6	RP-05-02	Reports	Summary Report	Generate at domain level			Login as domain administrator. Only displays events for the selected domain (no sub-domains).										
7	RP-05-03-01	Reports	Summary Report	Export report	Export as pdf												
8	RP-05-03-02	Reports	Summary Report	Export report	Export as spreadsheet												
9	RP-05-03-03	Reports	Summary Report	Export report	Print		Printer currently unavailable	!	!	!	!						

Figure 3: Touch Point Tool Example

or risk intensive [10].

To be most effective, the touch point abstraction layer needs to be incorporated into an integrated test tool suite design to provide automatic tracking of the associations between issues, functional requirements, interface points, and test cases (including automated scripts). The creation and maintenance of touch points should be controlled from the test team as a central part of the test basis, with tracking of touch point changes (and thus test cases) to approved issues (either defects or change requests). An example touch point spreadsheet (Figure 3) illustrates the use of historical results to provide a pattern view of coverage over builds, giving the tester a powerful tool to aid in exploratory or checklist tests. Finally, test reports need to be developed that indicate both touch point coverage as well as partial requirement coverage statistics.

Summary

The implementation of a touch point abstraction layer in the test basis would provide

1. an effective way to track partial requirement verification in the test domain space for complex requirements,
2. a means to help track implementation changes from delivery to delivery that may have an impact on areas such as training materials and user help documentation, and
3. a means to quickly document software changes when detailed requirement analysis is difficult or impossible to perform.

Although it is possible to perform this tracking separately, this abstraction layer would be most effective if implemented as a part of an integrated test management tool set. Such an addition to the tools would provide benefits in separating the complicated association of requirements to implemented designs derived during the test analysis and planning stages from the actual test implementation and execution.

References

[1] Gary E. Mogyorodi, "Requirements-Based Testing – Ambiguity Reviews", *Testing Experience Magazine*, number 2, pp. 28-30, March 2008.

[2] Heiko Köppen, "Traceable Knowledge by enhancing Requirements Management", *Testing Experience Magazine*, number 3, pp. 53-56, June 2008.

[3] Erik Runhaar, "Mind the gap between Test Strategy and Execution", *Testing Experience Mag.*, number 2, pp. 80-81, June 2008.

[4] An example "UI mapping" capability can be found with the SQA Design Qualify test tool at http://sqadesign.com/test-case-management/ui-mapping.html

[5] Rex Black, "How Agile Methodologies Challenge Testing", *Testing Experience Mag.*, number 7, pp. 36-40, September 2009.

[6] Chetan Giridhar, "Advocating Code Review for Testers", *Testing Experience Mag.*, number 11, pp. 56-58, September 2010.

[7] Rex Black, "Surveying Test Strategies: A Guide to Smart Selection and Blending", *Testing Experience Mag.*, number 2, pp. 32-34, June 2008.

[8] Armin Beer, Michael Menzel, "Test automation patterns: Closing the gap between requirements and test", *Testing Experience Mag.*, number 4, pp. 69-72, December 2008.

[9] Gary E. Mgyorodi, "Requirements-Based Testing – Cause-Effect Graphing", *Testing Experience Mag.*, number 5, pp. 40-44, March 2009.

[10] Brenda Kerton, "Requirements Traceability: Why Bother", *Software Quality Connection*, January 14, 2011, http://www.softwarequalityconnection.com/2011/01/requirements-traceability-why-bother/

> biography

Jeffrey H. Lucas
is a Senior Software QA Engineer with SecureInfo Corporation, San Antonio, Texas, U.S.A. He holds a Certified Tester Advanced Level certification through the ISTQB, has eight years of experience in software testing, and is the test lead on a small product development team. Prior to that, he has over ten years of test automation experience in other industries. His current interests are in the development of software test practices suitable for small development team environments.

Online Training

Díaz Hilterscheid

ISTQB® Certified Tester Foundation Level (English & German)
ISTQB® Certified Tester Advanced Level - Test Manager (English)
ISTQB® Certified Tester Advanced Level - Test Analyst (English)
ISTQB® Certified Tester Advanced Level - Technical Test Analyst (English)
ISEB Intermediate Certificate in Software Testing (English)

iseb Qualifications from BCS

ISTQB® Accredited Training Provider

ISTQB® International Software Testing Qualifications Board

Díaz Hilterscheid

te testing experience

Educating new hire as a part of improving testing process

by Lilia

When we hear "Improving of the Testing Process", we often imagine the desert (= weak testing process) and the relief provided by rain (= improving the process).

However, if we stop thinking in such manner, we could draw quite a different picture: flowers (= testing process) and everyday showers (= improving the process). The main idea is to feel the testing process and correct it before it is destroyed. Is this better?

As part of such an approach, I propose to take a look at educating new employees as a part of improving the testing process. How much time does it take to educate a new project member? You don't know? This may be because the process of educating a new employee is undefined and unmeasured. Just look around: Do you have a straight-forward process description for the new employee? The usual answer is that they are given documents/help/presentations. This is not an educational approach. It will take a lot of time from both mentor and new employee. The poor new employee will sink in an ocean of information.

To decrease the education time needed and increase efficiency, I propose to set up an educating system with the following steps. The system, of course, should be modified in accordance with company policy.

⇨ **Pre-requisites:** Each new employee should have a mentor – a person who will help to adapt in a new office, new project and redirect, if necessary, any requests.

Welcome tour of the office: Yes, I mean it, this is not a joke! It is important to show the area and to make the first steps in communicating and breaking the ice. If you cannot easily find your way in the office, you cannot feel free and productive. This easy rule is often forgotten during the educating process of new employees.

The list of the first steps and contacts. It could be a printed version, an e-mail or a corporate Wiki-page. What should be in this list? Everything, because some self-evident actions could be problematic for the new employee. For example, if the company has a helpdesk service, you as the mentor should provide the link and a quick guide on how to use it.

! Every time you add a detailed instruction or a guide to the list, you decrease the time that a mentor has to spend with a new employee.

! New employees cannot remember all the information they get at the start, so refreshing newly obtained knowledge via guides and short instructions is a good idea. Don't forget that new employees may feel uncomfortable, because of the new position and the new environment and that they may hesitate to ask something again and again.

Let's go back to the list of the first steps and contacts. It should contain:

Company and department overview

The list of the corporate resources should be available for the new employee. I want to emphasize this point. It is a common situation: the new employee does not have access to share "ABC", but nobody remembers how to obtain the access. The mentor starts to ask the project members: "Maybe somebody remembers this stuff??? NOBODY??? Ok, let's just write just to the helpdesk... and to Sam, Dan, Anna... to everybody." The never-ending investigation process "Who is responsible for this" starts. If it is a small company – it's clear, but if it is not? Just think for a minute how much time you could save with a simple instruction that is reusable. The best way to keep such instructions up to date is a Wiki page. In case of any changes, the new employee can modify the instructions.

Contacts. Every area has a person responsible for it. It is important to have a list of these persons. The situation can become critical when for example your main software\server goes down and stops all your work. Immediate contact of the responsible person would then be helpful.

⇨ The list of software for internal\project use. There could be internal software for task tracking, bug-tracking, source control etc. It is important to understand what instruments should be used in everyday work.

- **Policies.** We often don't like policies – hundreds or maybe thousands of pages of rules. It is not really exciting to read,

but it is an important part of our work. It provides structure and prevents chaos. It is impossible to work in cooperation with other project team members without understanding the policies.

- **Project learning.** This is an important step that especially requires the mentor's attention. Before you start the learning process, give the new employee a simple table that will help to collect important information of the "fresh look" owner.

It is more reasonable to give the full version of the table to a person that has enough experience in testing. In other cases, the table could be modified.

Item	Comments
Process\ What was new for you?	
Process\ What do you like?	
Process\ What do you suggest to improve?	
Project\ Specification	
Project\ Help	
Project\Usability	
Project\Functional	

Form for the new employee

Ask the new employee to make comments in the table during project learning. Then save the form and try to collect forms in future.

Also, you can use this information with project post-mortem documents when planning new versions of the product. Just think about it: When a new project version starts, you will have no time to stop and remember all the stuff that you want to improve or modify in the new version. Now you won't have to remember it! You just take all the forms, select the data and set the priorities. You start your strategic thinking.

Applying the strategy of a defined new employee learning process and getting early feedback from the new employee could give 3 benefits:

- Decrease learning period for new hire
- Decrease required mentor's time
- Collect data for new future releases

And the last benefit? It will also help you understand the level of experience of the new employee and find ways of better communication.

> biography

Lilia Gorbachik
is a QA team lead at Kofax, in their Business Process Automation department, in Saint-Petersburg, Russia. She has enjoyed working in software development. She has experience in product and outsourcing companies. Lilia contributes articles about testing to Russian testing communities. She has participated in the SQA Days conference that is the most important testing conference in Russia.

Can agile be certified?

Find out what Aitor, Erik or Nitin think about the certification at **www.agile-tester.org**

Training Concept

All Days: Daily Scrum and Soft Skills Assessment

Day 1: History and Terminology: Agile Manifesto, Principles and Methods

Day 2: Planning and Requirements

Day 3: Testing and Retrospectives

Day 4: Test Driven Development, Test Automation and Non-Functional

Day 5: Practical Assessment and Written Exam

Certified Agile Tester

We are well aware that agile team members shy away from standardized trainings and exams as they seem to be opposing the agile philosophy. However, agile projects are no free agents; they need structure and discipline as well as a common language and methods. Since the individuals in a team are the key element of agile projects, they heavily rely on a consensus on their daily work methods to be successful.

All the above was considered during the long and careful process of developing a certification framework that is agile and not static. The exam to certify the tester also had to capture the essential skills for agile cooperation. Hence a whole new approach was developed together with the experienced input of a number of renowned industry partners.

Supported by

Barclays
DORMA
Hewlett Packard
IBM
IVV
Logic Studio
Microfocus
Microsoft
Mobile.de
Nokia
NTS
Océ
SAP
Sogeti
SWIFT
T-Systems Multimedia Solutions
XING
Zurich

Is a Tester Responsible for Quality?

by Mikhail Pavlov, Ph.D.

In one of the popular blogs for project managers I read for the umpteenth time that a tester is responsible for quality. The number of such allegations has become critical, and I would like to remind stakeholders about what exactly a tester is responsible for in a project, or, more generally, what a testing group of a whole project is responsible for.

Definition of quality

One way or another, everyone has their own understanding of what quality is. My experience is mainly associated with working in companies that develop custom (outsourced) software. That is why I prefer the following definition of quality — which is product compliance with customer's requirements. Obviously, this is not a single definition of quality, and perhaps not the most accurate one either. Nevertheless, when I use the word „quality" in this article, I'm referring to the definition given above.

A tester is not responsible for the quality of a software product

My daughter asked me to check her homework in algebra. When I checked her work, I discovered a few mistakes. Moreover, I quite clearly noted where and why they were made. My daughter took her work back, and the next day she reported: „Dad, thanks to you I've got a low grade." It turned out that the day before my daughter had been rather lazy and had not corrected the mistakes that I had pointed out to her. So today she tried to place her responsibility for ignored mistakes on the person who had reported about them.

In the same way, a tester can during development of a project be accused of finding „too many" errors or „wrong errors", or of finding them „at the wrong time". Then a project manager decides that he is not going to correct all of the errors (or is not going to correct any errors at all), and there is a high chance that these errors will be found either by a customer during acceptance testing, or by an end-user during operation.

Thus, *a tester typically cannot influence the decision about whether or not to fix inadequacies of a product (defects) and, consequently, the quality of a product cannot fall into his responsibility area.*

In this regard, a project manager's face won't shine with happiness, because it is clear that incompetent or unsuccessful project managers might want to consider everyone, and especially a testing group, as their crime buddies. If they manage to pass on their responsibility for the unsuccessful project – that will be even better.

However, a project manager (and not a team or testers) is usually a team member who is solely responsible for quality of a product, because he/she is the only person who makes the key project decisions, including the decision to delegate some of her power.

Responsibilities of a tester

Obviously, if testers are not responsible for quality, while their services are still required for the project, they are responsible for something else.

A tester provides testing services to a development team. A development team, provided they have a proper understanding of the responsibilities in the project, expect from a tester to deliver current and accurate information about the current state of a software product, as well as forecasting the success of its development in terms of detection of bugs and their correction.

- What is the qualitative and quantitative assessment of the current state of the product in terms of its compliance to customer requirements?
- Will the project team be able to deliver the product on time and in good quality, if the current trends of detecting and correcting defects continue as it is?
- If a prognosis is unfavourable, then what kind of corrective measures are recommended to be taken?

If, at any point, a tester can provide to the stakeholders answers to at least the first two questions, then he deservedly earns his bread and butter. And if he can answer all three questions, then he, with a clear conscience, can top his buttered bread with a thin layer of caviar.

Reasons of misconceptions about testers' responsibilities

The first reason is an established practice of mixing two things together (meaning software testing and quality assurance). This occurs quite often and came together with the term of ‚quality assurance'.

We will probably never find out whose inflamed brain first decided to call testers ‚quality assurance experts'. Moreover, it is clear that some people, especially beginners, may find it more prestigious to be called ‚quality assurance experts', rather than ‚software testers'. As before, websites of employers and recruitment agencies are full of advertisements seeking QA-engineers and QA-managers [1], and their list of responsibilities includes duties of a typical tester/test designer and a typical test manager. This is clear evidence that employers have formed incorrect expectations towards testers, who they loudly (and with obvious pleasure) call ‚quality assurance experts'. Therefore, the scepticism of professional testers reading such ads is quite justified.

In fact, as we have already explained, testers are not responsible for the quality, because usually they cannot influence decisions on the quality of a product.

In addition, Michael Bolton noted in his well-titled blog post "Testers: Get Out of the Quality Assurance Business" [2],that testers are not involved in programming and cannot make changes to a code, that is they cannot directly improve the quality of a software code. Commenting on work of testers, M. Bolton wrote that it is not quality assurance, but **quality assistance**.

The second reason has already been discussed. Managers and other participants of a project are often ready to shift the responsibility for flaws onto testers, which is why they feel comfortable if a testing group labors under a misapprehension that they are responsible for product quality.

The third reason is that executives sometimes have a sincere misapprehension that testers are able to assure quality. The author of this article was faced with situations when executives decided to organize a systematic testing at organizational or project level, primarily because they believed that testers would assure quality.

Instead of a conclusion

The responsibilities of a tester (or testing group) are one of the key questions, and all stakeholders of the project should know the answer to this question, because this determines the success of the project. If stakeholders' expectations coincide with the objectives of a testing group, then the interaction and communication of testers and developers goes smooth, without serious conflicts, and stakeholders at different levels receive timely information about project status and know how to use this information.

Of course, success of a project cannot be simply achieved by a shared understanding of the responsibilities of a testing group, but everyone should understand that this condition must be a prerequisite.

---

[1] The question of who QA-engineer and QA-manager are is a topic for another discussion. Here I'd only like to note that quality assurance experts are responsible for monitoring the adherence to the software development processes, but not for the quality of a software product.

> biography

Mikhail Pavlov, Ph.D.

I have been working in the area of software testing and quality assurance for 15 years. My subjects of interest during the year of post-graduate studies were also standardization of programming languages and their standards conformance control. This interest was materialized in my Ph.D. thesis in 1994.

In the role of test manager at Luxoft I completed a dozen of international projects including a huge mainframe system maintenance project for avionics.

From 2001 I became involved in software process improvement activities in all companies I worked for. I was a SEPG manager in a CMMI level 4 certified company named Auriga for almost 3 years. Along with this role I was a head of a small training department and was responsible for the company's training system support.

In September 2009 I re-joined Luxoft in the role of Quality manager. The company-wide quality management system and project audits and measurements are under my supervision. Also I support continuity of ISO9001:2008 and CMMI level 5 certification across the company. Along with these responsibilities I am involved in the software testing community activities.

Improving the quality assurance process through beta testing

by Artur Gula

Every process improvement is difficult if you do not recognize incorrectness and find the causes.

On a sequential production line, for example when one stage is defective, this can be discovered during another step, which triggers it to begin the necessary improvements.

The same can be said for software development where the problems are usually found by the QA team and reported as an increased amount of bugs, delays in system delivery etc. There can be a few reasons for such a situation, but having continuous feedback from the QA allows project managers to pinpoint the origin of the problems and implement corrective actions, like additional team training or technological upgrades.

System or acceptance testers used to usually perform the last formal product verification. In this case, test managers usually do not get enough external feedback. There are, of course, some helpful tracking metrics, but when someone asks: "How good is your testing?", it is still quite difficult to find the right answer.

One effective method of internal test process evaluation is beta testing. Most people, when hearing "beta" think of final product rating, which in most cases is true. However, if beta testing is well organized and managed, it can also provide a lot of information about the QA team's work.

Managed beta testing

So far, we have found out "what" beta testing is, but the more important question to ask is: "How?"

Beta test management is such an extensive topic, that it couldn't be described fully even in a very detailed article. However, I'll try to specify the most important issues, which are necessary in order to get results at an appropriate level.

- Selection of testers. There are two main methods – quantitative and qualitative. The first one means that the application is distributed among hundreds or thousands of potential testers, the second one needs only few well prepared and motivated testers. The qualitative selection of testers is usually more effective and profitable.
- Test management. The beta test process, as with any other test activity, needs to be planned and managed correctly. There is need to schedule it, set entry and exit criteria etc. It is also necessary to establish the data flow between testers and beta engineers. Every incident report should be reviewed and then, according to the review results, additional actions must be taken.

  Also important is that every tester needs to be listened to, supported and appreciated, especially when the required criteria have been fulfilled.
- The whole beta process must be controlled and measured. Most importantly is the need to track the number of issues, their severity, their type (bug, proposal, usability issue etc.) and area in which they were found.

Beta testing cost

The next question to be asked is "How much does it cost?" Beta tests, although useful, should not increase the testing budget significantly. Fortunately, it usually costs much less than the system or acceptance test. The main reason for this is that beta testers are usually volunteers, who just want to try a new version of the software. It is recommended to offer some incentive program to motivate them, but they should not be rewarded with any kind of salary. Promising free copies of the checked application should be sufficient. Of course, this only applies for the testers who cooperated during the whole process.

Internal costs, like management, depend on the scale of the project. For smaller ones, a test manager can create a master test plan, then designate one or two engineers to support the participants. In the case of larger projects, it is necessary to create an additional beta section, which would conduct this process.

Benefits from beta

At this stage, we are starting to become familiar with the managed beta test process.

Let's take a look at some of the main advantages:

- If it is done right, it can provide a lot of information about our final product, which can be used to evaluate the QA team, their work and the whole quality assurance process. I'll give some practical examples in the last section of the article.

- Beta helps to improve the final product – this is obvious. Final users, who can be domain experts, find different types of malfunctions which couldn't be found during the previous testing process. They are also not biased in any way, which makes their point of view different from that of the QA team members.
- Costs – Although beta is still a part of the whole quality control process, it costs much less than any other activity, so its benefit/cost ratio is relatively high.
- Beta implementation increases the whole system reliability, because of its high level of independence.
- It is highly versatile - information gathered during beta can also be useful for other departments and not just the QA team, for example for human resources (when looking for potential candidates) or for marketing (when launching market research).

Beta risks

The description wouldn't be complete without risk analysis. There are a few core areas in which something could go wrong:

- Testers – As previously mentioned, the testers are volunteers who are motivated to participate in our project in some way . Because of this, we need to be prepared that some of them will not finish the testing process. There can be different reasons, like lack of free time, illness, personal matters which distract from testing, or just lack of motivation.
- We should always track our participants' level of involvement and motivation, and when it is reduced – try to increase it. A friendly and appreciative email or a promise of a reward can be helpful. If nothing can be done – all we need to do is to make sure all formalities are done (getting the application back from the tester etc.).
- Version problems – Delivering application stability is critical for the whole process. If the system under test has a lot of bugs which block the main functionality, it makes beta tests impossible to conduct, causing testers to become de-motivated, and stops them from further activity.
- Time issues – Beta tests are the last activity of the whole development process. Using management terms, it can be said that they are always on a "critical path". Any previous stage can easily handle a little delay, but there can be a lot of pressure on beta test managers when beta tests are late, because

Beta test observation	Possible cause	Corrective action
Beta test engineers notice a large number of bugs found.	Poor internal quality assurance process.	There is need to investigate the test process in order to find its weaknesses, and then implement adequate preventive actions.
Beta test engineers notice that in most cases bugs are connected to the same functionality.	Tester or sub-team responsible for this module were not well prepared or neglected their work. or This functionality is critical for the whole system, which is why beta testers concentrate on it.	There is a need to check the quality of analysis documentation for this module in case it contains a lot of mistakes. It may be necessary to improve the initial documentation review process. If test basis is at the same level for all modules, there must be additional training requested for this group of testers. Should the situation continue, it will show the attitude of testers toward their duties. or We need to launch risk based testing technique, in order to concentrate on the most important issues.
Beta testers are not reporting critical or even medium-level bugs.	Well planned and managed test process, including confirmation and regression testing found most of the bugs.	There is no need to implement changes, but such information can increase our product confidence which is important in critical business decisions. We still shouldn't forget that incorrectly conducted beta test processes could also bring no useful results.
Beta testers mainly complain about software usability.	QA team members are too "technical", and have lost the user point of view.	Software usability training should be helpful. or Usability tests should be outsourced.
Beta testers report a lot of critical bugs in complicated functionalities.	QA team is not familiar enough with the application.	Experienced application user should perform additional training. Analysis documentation should be more precise and legible. Scheduling a documentation review meeting is also recommended.
Beta testers complain about increased amount of bugs (even for previously positively tested areas).	No regression tests are run after major changes.	Regression tests should be added to the test plan – whether they are automated or manual.

the whole concern will be focused on him/her and the beta test team. What is more, it is difficult to accelerate beta activities, so it leads to stressful situations, for which we need to be prepared.

- Results acceptance – If serious issues are discovered during beta tests, they must be corrected and checked again. There may also be a need to perform some regression tests. Although this takes time, it may cause the detection of additional bugs.

At this stage in the project, all the above issues are critical issues and are not welcome. On the other hand, issues found by beta testers are usually more business-connected than technical, which makes it difficult to persuade the project manager to fix them. This can result in a difficult position for the test manager who should remain calm and assertive. Should a conflict occur that cannot be resolved, a triage meeting with stakeholders can be helpful.

Examples

As mentioned earlier, not only does beta testing help in software evaluation, but it also helps with internal quality assurance process judgment.

The table contains sample information obtained from the beta testing process and shows observations, possible causes and recommended corrective actions.

Summary:

- The beta test process is a reliable, inexpensive way to internally conduct test process evaluation.
- It improves product quality.
- Data gathered during beta can be useful in many ways by different departments.
- The beta testing must be well prepared and managed.
- There are serious project risks connected to it.

> biography

Artur Gula

I have been in software testing since 2008 when I passed the ISTQB® Foundation Level exam and started working as a tester in the medical field. Presently, I am a Test Manager at LGBS Software in Poland. I am mainly interested in test process management and test process improvement through different metrics, automation and additional activities, like beta testing.

German Testing Board

www.german-testing-board.info

ISTQB® Certified Tester

Über 165.000 Certified Tester weltweit. Wann gehören Sie dazu?

Manche Ausbildungen öffnen Wege, andere eine ganze Welt.

Die Schulung zum ISTQB® Certified Tester führt zu einem weltweit anerkannten und etablierten Zertifikat. Rund 17.700 Fachkräfte alleine in Deutschland haben es schon. Und Zehntausende in weiteren 47 Ländern - von A wie Amerika bis V wie Vietnam.

- Der ISTQB® Certified Tester ermöglicht Karrieren - mit ihm liegt erstmals ein internationales und branchenübergreifendes Berufsprofil für Software-Tester vor.
- Der ISTQB® Certified Tester macht die Arbeit leichter - Tester sprechen nun die gleiche Fachsprache, benutzen die gleichen Begriffe.
- Der ISTQB® Certified Tester hilft Kosten zu senken - Durch geschulte Tester werden Fehler bereits in frühen Phasen der SW-Entwicklung entdeckt.

Anmelden ist einfach.

Ein akkreditierter Trainingsanbieter ist sicher auch in Ihrer Nähe:

GTB Premium Partner

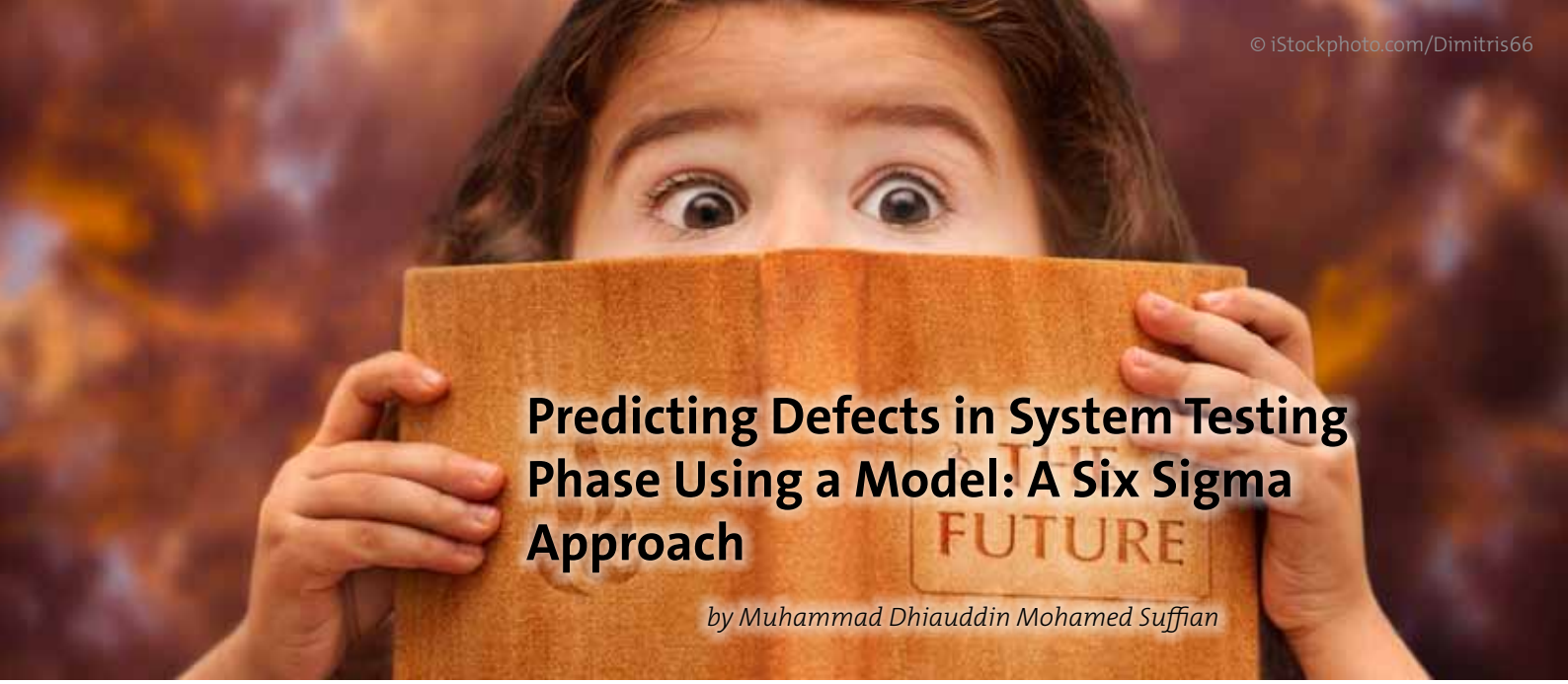
- ALTRAN GmbH & Co. KG
- andagon GmbH
- berner & mattner Systemtechnik GmbH
- corporate quality consulting GmbH
- EXCO GmbH
- imbus AG
- Knowledge Department GmbH & Co. KG
- Logica Deutschland GmbH & Co. KG
- MaibornWolff et al GmbH
- Method Park Software AG
- oose Innovative Informatik GmbH
- Philotech GmbH
- qme Software GmbH
- sepp.med gmbh
- Software Quality Lab GmbH
- Sogeti Deutschland GmbH
- SQS AG
- T-Systems

autorisierte Zertifizierungsstellen

DLGI - Dienstleistungsgesellschaft für Informatik mbH

iSQI - International Software Quality Institute GmbH

Alle Trainingsprovider siehe www.german-testing-board.info



Predicting Defects in System Testing Phase Using a Model: A Six Sigma Approach

by Muhammad Dhiauddin Mohamed Suffian

This article is a revised version of a paper presented at the 4th International Symposium on Information Technology (ITSIM'10)

Defect prediction is an important aspect of the software development life cycle. The rationale in knowing the predicted number of functional defects earlier on in the lifecycle, particularly at the start of the testing phase, is, apart from just find as many defects as possible during the testing phase, to determine when to stop testing and ensure all defects have been found in testing before the software is delivered to the intended end user. It also ensures that wider test coverage is put in place to discover the predicted defects. This research is aimed at achieving zero known post-release defects of the software delivered to the end user. To achieve the target, the research effort focuses on establishing a test defect prediction model using the Design for Six Sigma methodology in a controlled environment, where all factors contributing to defects in the software are collected prior to commencing the testing phase. It identifies the requirements for the prediction model and how the model can benefit from them. It also outlines the possible predictors associated with defect discovery in the testing phase. The proposed test defect prediction model is demonstrated via multiple regression analysis incorporating testing metrics and development-related metrics as the predictors.

Why use defect prediction at the start of the testing phase

1. *Better resource planning for test execution across projects*: As one test resource can work in more than one testing project, the prediction of defects in the testing phase allows for planning the appropriate number of resources to be assigned across multiple projects and support resource planning activities to ensure the optimized resources and higher productivity.
2. *Issue on wider test coverage in discovering defects*: To ensure wider and better test coverage, prediction of defects could improve the way of finding defects since there is now a target number of defects to be found, either by adding more types of testing or adding more scenarios related to user experience which results in better root cause analysis.
3. *Issue on improving test execution time to meet project deadline*: Putting defect prediction in place will reduce schedule slippage problems resulting from testing activities. By having a predicted number of defects to be discovered, test execution can be planned accordingly to meet the project deadline.
4. *Model also deals with issue on reliability of software to be delivered*: More defects found within the testing phase means more defects are prevented from escaping to the field. Having defect prediction provides a direction on how many defects should be contained within the phase and later, contributes to the zero known post-release defects of the software. In the long run, it shall portray the stability of the whole team effort in producing software.

Research Objectives

The main question to address is "How to predict the total number of defects to be found at the start of the system testing phase using a model?" This is followed by the list of sub-questions to support the main research questions as below:

- What are the key contributors to the test defect prediction model?
- What are the factors that contribute to finding defects in the system testing phase?
- How can we measure the relationship between the defect factors and the total number of defects in the system testing phase?
- What type of data should be gathered and how to get them?
- How can the prediction model help in improving the testing process and improve software quality?

From the list of questions, the research effort has been able to demonstrate the following key items:

- Establish a defect prediction model for software testing phase
- Demonstrate the approach in building a defect prediction model using the Design for Six Sigma (DfSS) Methodology
- Identify the significant factors that contribute to a reliable defect prediction model
- Determine the importance of a defect prediction model for improving the testing process

Building the Model

In Design for Six Sigma (DfSS), the research is organized according to DMADV phases: Define (D), Measure (M), Analyze (A), Design (D) and Verify (V). DfSS seeks to sequence proper tools and techniques to design in more value during new product development, while creating and using higher quality data for key development decisions. Achievement of DfSS is observed when the products or services developed meet customer needs rather than competing alternatives. In general, DfSS-DMADV phases are described as below:

- Define - identify the project goals and customer (internal and external) requirements
- Measure - determine customer needs and specifications; benchmark competitors and industry
- Analyze – study and evaluate the process options to meet customer needs
- Design – detailing the process to meet customer needs
- Verify – confirm and prove the design performance and ability to meet customer needs

A. Define Phase

The Define Phase primarily involves establishing the project definition and acknowledging the fundamental needs of the research. A typical software production process which applies the V-Model software development life cycle is used as the basis for this research. Throughout the process, the testing team is involved in all review sessions for each phase, starting from planning until the end of the system integration testing phase. The test lab is involved in reviewing the planning document, the requirement analysis document, the design document, the test planning document and the test cases (see Figure 1). The software production process is governed by project management, quality management, configuration and change management, integral and support as well as process improvement initiatives, in compliance to CMMI®. From the figure, the area of study is the functional or system test phase, where defects are going to be predicted. Therefore, only potential factors/predictors in the phases prior to the system testing phase are considered and investigated.

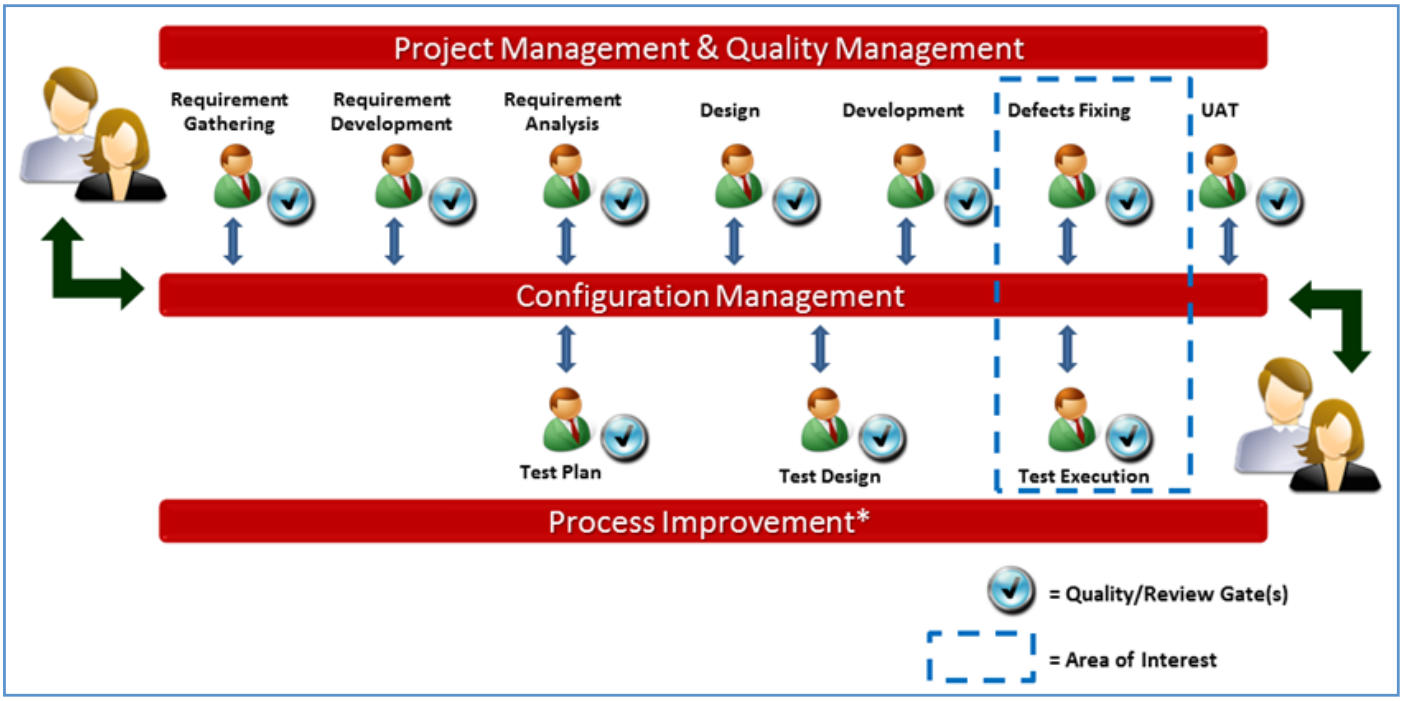


Figure 1: Typical Software Production Process

Two (2) schematic diagrams are produced, which are a high level schematic diagram (Figure 2) and a detailed schematic diagram (Figure 3). The high level schematic diagram deals with establishing the Big Y or business target, little Ys, vital Xs and the goal statement against the business scorecard. In this research, Big Y is to produce software with zero-known post-release defects, while for little Ys, elements that contribute to achieving the Big Y are defect containment in the test phase, customer satisfaction, quality of the process being imposed to produce the software and project management. There are two aspects involved related to these little Ys: potential number of defects before the test phase which is the research interest, and the number of defects after completing the test phase.

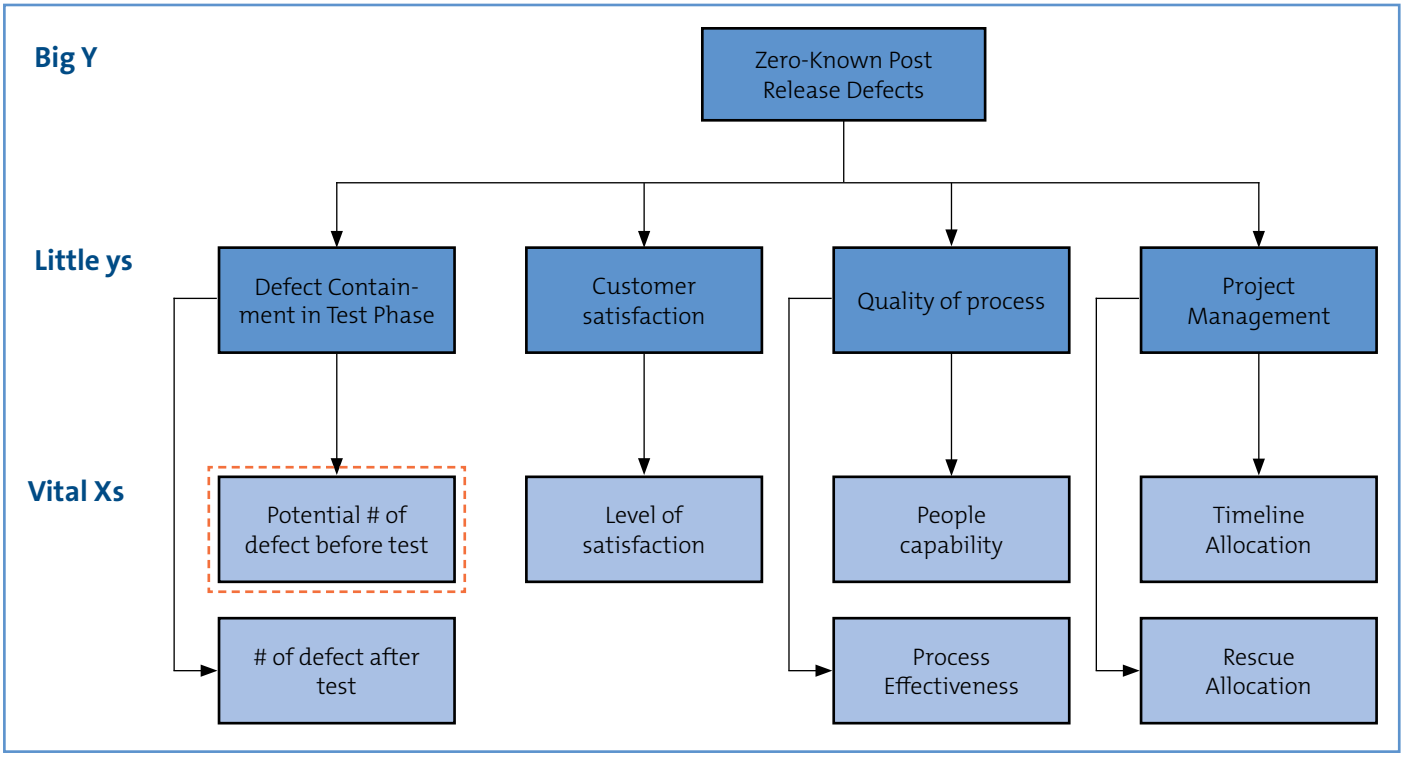


Figure 2: Schematic Diagram

For the detailed schematic diagram (or detailed Y-X tree), possible factors that contribute to the test defect prediction are defined from the Vital X, which is the potential number of defects before test, and these are summarized in a Y to X tree diagram as shown in the figure below. The highlighted factors are selected for further analysis.

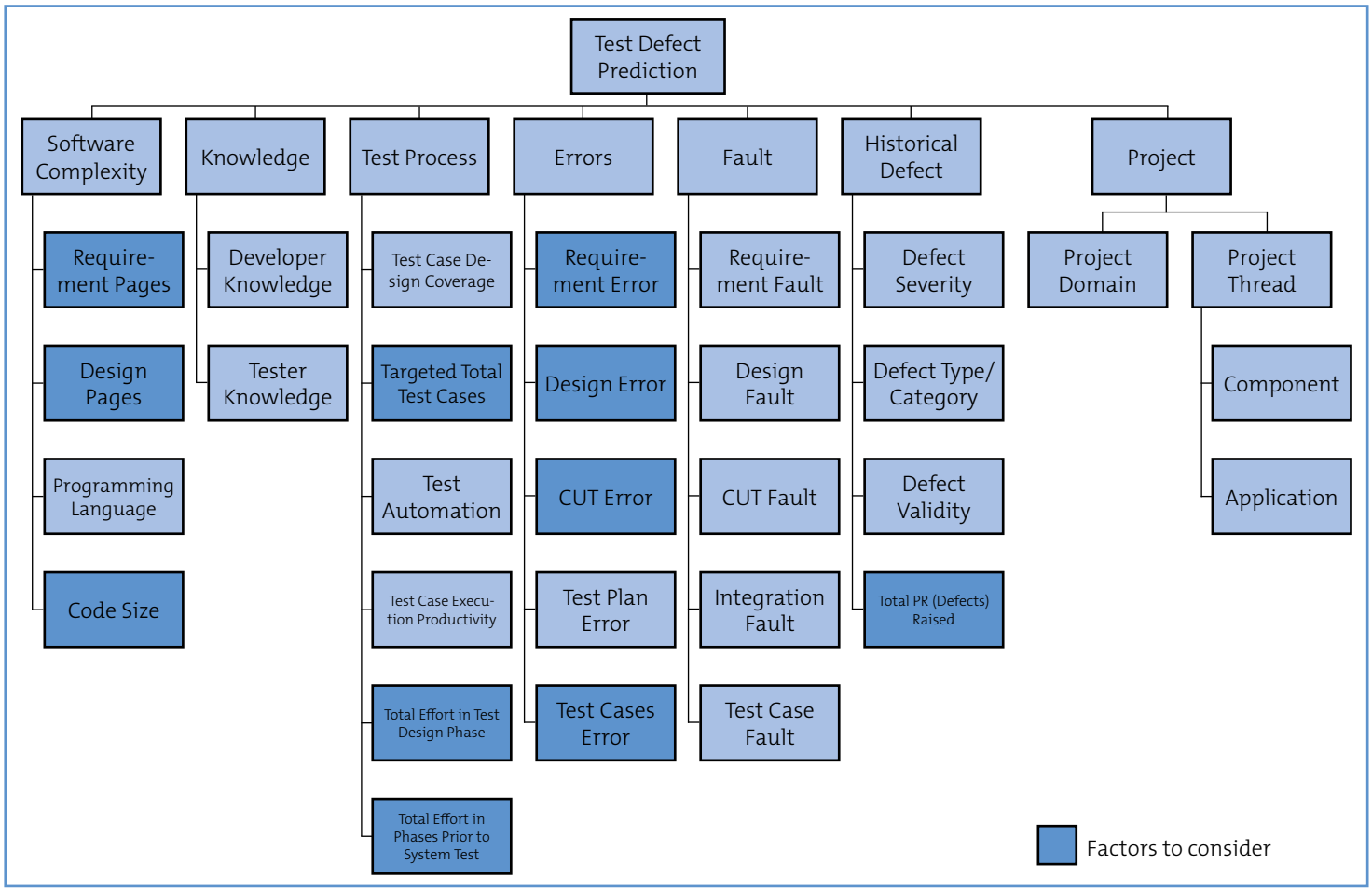


Figure 3: Detailed Y-X Tree Diagram

B. Measure Phase

A Measurement System Analysis (MSA) is conducted to validate that the processes of discovering defects are repeatable and reproducible, thus eliminating human errors. Ten test cases with known results of PASS and FAIL are identified. Then three test engineers are selected to execute the test cases at random. This is repeated three times for every engineer. The outcome is presented in Figure 4 below:

TCD ID	TC Result	Tester 1			Tester 2			Tester 3		
		1	2	3	1	2	3	1	2	3
TC1	PASS	PASS	PASS	PASS	PASS	PASS	PASS	PASS	PASS	PASS
TC2	FAIL	FAIL	FAIL	FAIL	FAIL	FAIL	FAIL	PASS	PASS	PASS
TC3	FAIL	FAIL	FAIL	FAIL	FAIL	FAIL	FAIL	PASS	PASS	PASS
TC4	PASS	FAIL	FAIL	FAIL	FAIL	FAIL	FAIL	PASS	PASS	PASS
TC5	PASS	PASS	PASS	PASS	PASS	PASS	PASS	PASS	PASS	PASS
TC6	PASS	PASS	PASS	PASS	PASS	PASS	PASS	PASS	PASS	PASS
TC7	PASS	PASS	PASS	PASS	PASS	PASS	PASS	PASS	PASS	PASS
TC8	FAIL	FAIL	FAIL	FAIL	FAIL	FAIL	FAIL	FAIL	FAIL	FAIL
TC9	PASS	PASS	PASS	PASS	PASS	PASS	PASS	PASS	PASS	PASS
TC10	PASS	PASS	PASS	PASS	PASS	PASS	PASS	PASS	PASS	PASS

Figure 4: Data for Analysis

The data is then used to run the MSA, where the result is compared against the Kappa value. Since the overall result of all appraisers against standard is above 70% (as required by Kappa's value), the MSA is considered as a PASS as presented in Figure 5 below:

As the MSA is passed, the operational definition is prepared consisting of type of data to be gathered, measurement to be used, responsibilities, and mechanism to obtain those data so that the data can be used for next DfSS phase.

All Appraisers vs Standard

Assessment Agreement

# Inspected	# Matched	Percent	95 % CI
10	8	80.00	(44.39, 97.48)

Matched: All appraisers' assessments agree with the known standard.

Fleiss' Kappa Statistics

Response	Kappa	SE Kappa	Z	P(vs > 0)
FAIL	0.764591	0.105409	7.25355	0.0000
PASS	0.764591	0.105409	7.25355	0.0000

Overall Result for MSA is PASS since Kappa value is greater than 0.7 or 70%

Figure 5: Overall Result of MSA

C. Analyze Phase

The data gathered from the Measure Phase is used to perform a first round of regression analysis using the data shown in Figure 6, which was collected from thirteen projects. The predictors used include requirement error, design error, Code and Unit Test (CUT) error, Kilo Lines of Code (KLOC) size, targeted total test cases to be executed, test plan error, test cases error, automation percentage, test effort in days, and test execution productivity per staff day. The regression is done against the functional defects as the target. MINITAB software is used to perform the regression analysis via manual stepwise regression analysis, i.e. the predictors are added and removed during the regression until a strong statistical result is obtained. The figure involves the P-value of each predictor against the defects, the R-squared (R-Sq.) value and the R-squared adjusted (R-Sq. (adj.)) value. These figures demonstrated the goodness of the equation and how well it can be used for predicting the defects. The result of the regression analysis is presented in Figure 7.

Project Name	Req. Error	Design Error	CUT Error	KLOC	Total Test Cases	Test Plan Error	Test Case Error	Automation %	Test Effort	Test Execution Productivity	Functional Defects
Project A	5	22	12	28.8	224	0	34	0	6.38	45.8000	19
Project B	0	0	1	6.8	17	0	6	0	9.36	17.0000	1
Project C	9	10	14	5.4	24	4	6	0	29.16	5.8333	4
Project D	7	12	2	1.1	25	4	9	0	13.17	7.0000	0
Project E	11	29	3	1.2	28	4	12	0	14.26	3.4000	3
Project F	0	2	7	6.8	66	1	7	0	32.64	31.0000	16
Project G	3	25	11	4	149	5	0	0	7.15	74.5000	3
Project H	4	9	2	0.2	24	4	0	0	18.78	7.6667	0
Project I	7	0	1	1.8	16	1	3	0	9.29	2.6818	1
Project J	1	7	2	2.1	20	1	4	0	6.73	1.9450	0
Project K	17	0	3	1.4	13	1	4	0	8.44	6.5000	1
Project L	3	0	0	1.3	20	1	7	0	14.18	9.7500	1
Project M	2	3	16	2.5	7	1	6	0	8.44	1.7500	0

Figure 6: Data for Regression Analysis

Regression Analysis: Functional_D versus Design Error, TotalTC_1, ...

The regression equation is

Functional_Defects_1 = - 3.04 + 0.220 Design Error_1 + 0.0624 TotalTC_1
- 2.30 TP Error_1 + 0.477 Test Effort_1

Predictor	Coef	SE Coef	T	P
Constant	-3.0414	0.9218	-3.30	0.011
Design Error_1	0.21970	0.07911	2.78	0.024
TotalTC_1	0.062379	0.009565	6.52	0.000
TP Error_1	-2.2964	0.3719	-6.18	0.000
Test Effort_1	0.47709	0.05345	8.93	0.000

S = 1.39691 R-Sq = 96.7% R-Sq(adj) = 95.0%

Figure 7: Regression Analysis Result

Project Name	Req. Error	Design Error	CUT Error	KLOC	Req. Page	Design Page	Total Test Cases	Test Cases Error	Total Effort	Test Design Effort	Functional Defects	All Defects
Project A	5	22	12	28.8	81	121	224	34	16.79	15.20	19	19
Project B	0	0	1	6.8	171	14	17	6	45.69	40.91	1	1
Project C	9	10	14	5.4	23	42	24	6	13.44	13.44	4	4
Project D	7	12	2	1.1	23	42	25	9	4.90	9.90	0	0
Project E	11	29	3	1.2	23	54	28	12	4.72	4.59	3	3
Project F	0	2	7	6.8	20	70	88	7	32.69	16.00	16	27
Project G	3	25	11	4	38	131	149	0	64.00	53.30	3	3
Project H	4	9	2	0.2	26	26	24	0	5.63	5.63	0	0
Project I	17	0	3	1.4	15	28	13	4	9.13	7.88	1	1
Project J	61	34	24	36	57	156	306	16	89.42	76.16	25	28
Project K	32	16	19	12.3	162	384	142	0	7.00	7.00	12	12
Project L	0	2	3	3.8	35	33	40	3	8.86	8.86	6	6
Project M	15	18	10	26.1	88	211	151	22	30.99	28.61	39	57
Project N	0	4	0	24.2	102	11	157	0	41.13	28.13	20	33

Figure 8: New Set of Data for Regression

Based on the result, it has been demonstrated that for this round of regression, possible predictors are design error, targeted total test cases to be executed, test plan error, and test effort in days, in which the P-value for each predictor is less than 0.05. As overall model equation, this model portrays strong characteristics of a good model via high percentage of R-Sq. and R-Sq. (adjusted) values: 96.7% and 95.0% respectively. From the regression, this equation is selected for study in the next phase:

Defect = - 3.04 + 0.220 Design Error + 0.0624 Targeted Total Test Cases - 2.30 Test Plan Error + 0.477 Test Efforts

D. Design Phase

Further analysis is conducted during the Design Phase due to the need to generate a prediction model that both is practical and makes sense from the viewpoint of software practitioners using logical predictors, to filter the metrics to contain only valid data, and to reduce the model to have only coefficients that have a logical correlation to the defect. As a result, a new set of data is used to refine the model as shown in Figure 8.

Using the new set of data, new regression results are presented in Figure 9.

Regression Analysis: Functional_Defects versus Req Error, CUT error, ...

```
The regression equation is
Functional_Defects = 4.00 - 0.204 Req Error - 0.631 CUT error + 1.90 KLOC
                     - 0.140 ReqPage + 0.125 DesPage - 0.169 TotalTC
                     + 0.221 EffortDays


Predictor       Coef  SE Coef      T      P
Constant      3.9969   0.9435   4.24  0.005
Req Error   -0.20389  0.05001  -4.08  0.007
CUT error    -0.6309   0.1566  -4.03  0.007
KLOC          1.9050   0.1249  15.25  0.000
ReqPage     -0.14043  0.01515  -9.27  0.000
DesPage      0.12459  0.01151  10.82  0.000
TotalTC     -0.16911  0.02055  -8.23  0.000
EffortDays   0.22127  0.03439   6.43  0.001


S = 1.80987   R-Sq = 98.9%   R-Sq(adj) = 97.6%
```

Figure 9: New Regression Result

From these latest results, it can be demonstrated that significant predictors for predicting defects are requirement error, CUT error, KLOC, requirement page, design page, targeted total test cases to be executed, and test effort in days from the phases prior to system test. The P-value for each factor is less than 0.05, while the R-Sq. and R-Sq. (adjusted) values are 98.9% and 97.6% respectively, which results in a stronger prediction equation. The selected equation is as below:

***Functional Defects (Y)** = 4.00 - 0.204 Requirement Error - 0.631 CUT error + 1.90 KLOC - 0.140 Requirement Page + 0.125 Design Page - 0.169 Total Test Cases + 0.221 Test Effort*

E. Verify Phase

The selected model in the Design Phase is now verified against new projects that have yet to go to the System Test phase. The actual defects found after the System Test has been completed are compared against the predicted defects to ensure that actual defects fall between 95% prediction intervals (PI) of the model as presented in the last column in Figure 10.

No	Predicted Functional Defects	Actual Functional Defects	95% CI (min, max)	95% PI (min, max)
1.	182	187	(155, 209)	(154, 209)
2.	6	1	(0, 2)	(0, 14)
3.	1	1	(0, 3)	(0, 6)

Figure 10: Verification Result for the Prediction Model

Based on the the verification result, it is clearly shown that the model is fit for use and can be implemented to predict test defects for the software product. This is justified by the predicted defects number, which falls within 95% Prediction Interval (PI). As the test defect prediction model equation is finalized, the next consideration is to emphasize on the control plan with regard to its implementation in the process, i.e. when the actual number of defects found is lower or greater than the prediction. Figure 11 below summarizes the control plan:

Actual Defects < Predicted Defects	Actual Defects > Predicted Defects
Perform thorough testing during ad-hoc test	Re-visit the errors captured during requirement, design and CUT phase
Perform additional test strategy that relates to the discovery of more functional defects	Re-visit the errors captured during test case design
	Re-visit the model and the factors used to define the model

Figure 11: Action Plan for Test Defect Prediction Model

Conclusion

From the findings in this research, it is proven that Six Sigma provides a structured and systematic way of building a defect prediction model for the testing phase. The Design for Six Sigma (DfSS) methodology provides opportunities to clearly determine what needs to be achieved from the research, issues to be addressed, data to be collected, the needs to be measured and how the model is generated, constructed and validated. Moreover, from the model equation it is possible to discover strong factors that contribute to the number of defects in the testing phase, while acknowledging the need to consider other important factors that have significant impact to on testing defects. This research has also demonstrated the importance of a defect prediction model in improving the testing process, since it contributes to zero-known post-release defects of a software. Having test defect prediction helps in better test strategy and wider test coverage, while at the same time ensuring stability of overall the software development effort for releasing a software product.

References

1. Graham, G., Veenendaal, E.V., Evans, I. and Black, R. (2006). Foundations of Software Testing: ISTQB Certification. Thomson Learning. United Kingdom.
2. Fenton, N.E. and Neil, M. (1999). A Critique of Software Defect Prediction Models. IEEE Transactions On Software Engineering. Volume 25, No.5.
3. Clark, B. and Zubrow, D. (2001). How Good is the Software: A Review of Defect Prediction Techniques. Software Engineering Symposium. Carnegie Mellon University.
4. Nayak, V. and Naidya, D. (2003). Defect Estimation Strategies. Patni Computer Systems Limited. Mumbai.
5. Thangarajan, M. and Biswas, B. (2002). Software Reliability Prediction Model. Tata Elxsi Whitepaper

> biography

Muhammad Dhiauddin Mohamed Suffian
is pursuing his PhD (Computer Science) in Software Testing at Universiti Teknologi Malaysia and working as lecturer at the leading open-university in Malaysia. He was the Solution Test Manager at one of the public-listed IT companies in Malaysia and prior to that, he was Senior Engineer and Test Team Lead for the test department in one of the leading R&D agencies in Malaysia. He has almost 7 years of experience in the software/system sevelopment and software testing / quality assurance fields. With working experience in IT, automotive, banking and research & development companies, he obtained his technical and management skills from various project profiles. A graduate of M.Sc. in Real Time Software Engineering from Centre for Advanced Software Engineering (CASE), Universiti Teknologi Malaysia, he holds various professional certifications, namely Certified Six Sigma Green Belt, Certified Tester Foundation Level (CTFL) and Certified Tester Advanced Level – Test Manager (CTAL-TM). He also has vast knowledge in CMMi, test process and methodologies as well as the Software Development Life Cycle (SDLC). He was involved in and has managed various testing strategies for different projects, including functional, performance, security, usability and compatibility test, both at system test and system integration test level. His interest falls within software engineering and software testing areas, particularly on performance testing and test management.

Key Questions to ask when improving testing

by Erwin Engelsma

When you decide to improve the testing process in your organization, it is essential that you follow a structured approach, as described in well-known improvement models like TPI® or TMMi®[1]. Just doing something because it seems to be the right thing will almost certainly be a waste of effort and money, at best.

To achieve a well-defined goal, there are some questions to ask yourself before starting improvement efforts. Just going through the motions of following the steps that the models describe won't do anyone any good.

So what are some of the things to really think about?

Read on ...

Contribution to your business

What contribution to your business would improving the testing process really make?

Several classes of answers are possible. You can summarize them as follows:

1. Better testing would somehow improve the end quality of your product, resulting in better customer satisfaction, fewer returned products and more sales;
2. Better testing would somehow shorten the development cycle of your product resulting in lower project costs and help towards a speedier market introduction; also, a better controlled project may be possible;
3. Better testing would somehow save on product costs as the total amount of effort might decrease;
4. Better testing would somehow bring you in line with external laws so you can supply products to new markets, hence increasing your sales;
5. Ah yes, and there is the adage that proper testing helps management form an informed and objective judgement about releasing the product or not.

While the above reasons may be valid in your particular circumstances, there are a number of further questions to ask, especially about the meaning of the ‚somehow' and further assumptions you may have made.

Almost like the serpent in paradise, you need to ask whether it is ‚really true' that better testing results in better quality of your product and better customer satisfaction.

First point:

Testing in itself does not improve anything. It is only when effective testing is part of a product improvement cycle, and after root causes analysis, changes to designs, code and improvement in developers' skills (and so on) are made, that testing contributes to this aim.

Testing can improve customer satisfaction if you actually know what the customer thinks is truly important and test for those subjects. Making great improvements to an area that your customer is hardly interested in is a laudable effort, but won't much increase their idea of how good your product is! If you want to improve customer satisfaction (how do you know what satisfies them? Do you really know what customers expect from your products and how they use them?), you would do better first to find out how your products are used, what customers are unhappy about, and what actual 'failure modes' for your products are.

And yes, you have to embed the testing effort in the development organization and make sure test results are analyzed and defects solved early.

Second point:

It is well known that a defect found in a late phase of product development (or after product completion) costs a lot more to solve than that same defect found at the beginning of a development cycle, during a review, unit test or some other test activity early in the development cycle. So it is obviously a good idea to start activities to find faults at an earlier stage. But ... do you know where those activities are the most effective? Should you start by creating better requirements, trace them better to test cases, improve and model designs, improve code reviews, do better unit testing, better integration testing, better functional testing, bet-

1 *Test Maturity Model (integration)*

ter reliability testing? Create user profiles, and automate their execution? Focus on mechanisms in the code that go consistently wrong (memory leaks, over-usage of processor capacity, other resource usage)? Better usability testing?

Unless you were really getting bored, there is no way you can improve all these activities at the same time. And for each of the items mentioned, you (your organization) must either have the skills or acquire them, and an opportunity to deploy them in your organization. Standing on the sideline is hardly a good idea; usually you use a project developing some product as a carrier to introduce the new skills. And there should not be much time between 'learning' a new skill and applying it in practice, or else the new skills will just evaporate.

Make strategic choices: plan for small increments in your improvements that show a result almost immediately. And there are always the little details that make everything a lot harder to do than you thought. Create a few unit tests in say 'N-Unit'? No problem, until you hit legacy code that is virtually impossible to test due to SW architecture that never took testing into account. Improve reviewing? No problem, until you find you are in a culture where people do not call a spade a spade, and rather than hurting each other's feelings take the risk of a product that is less good than it might be. There you go, thinking you would just improve some reviewing skills, and you find you need to change a complete culture and mentality. So the lesson is almost to run an FMEA[2] of your organization before introducing an improvement. Wonder what you want to improve, why this particular area, think of all the factors that may go wrong, face them, address these issues.

Third point:

This finds its justification in that defects you find early save rework later on. However, you must create the right circumstances before this becomes valid. You may find you run into so many unexpected problems inside your product that you would never have found otherwise that to start with you create more work on the whole. Now is the time not to give up and to keep on analyzing what was different from what you expected and why. By the way: have you thought of a way of measuring results so that management is convinced?

Fourth point:

If some legislative body like the FDA[3] is essential to your product sales, there is only one piece of advice: make sure you fulfil their demands. Do it in a wise way. Hire an expert on their rules and regulations if needed, but satisfy their demands.

Fifth point:

Yes, you may provide your management with better information, but can they handle it? And are you capable of providing them with information that is relevant enough for the business to even be worth their attention? Find out their needs and expectations. See how you can link into that. See to what extent you can change them if that is really needed.

Ponder these questions and make sure their solutions are part of the test policy you write. How are you going to make this understood by all the people involved? Note that as you are the one who is very motivated and knowledgeable about the subject, you may take a lot of knowledge and practices for granted. In fact, many things you believe are not self-evident. So really put yourself in the position of the people you are trying to convince, understand their hesitations and have a really good answer to them.

Where are you now?

Both TPI® and TMMi® have excellent questions to help you decide the maturity level of your organization. Doing an assessment or having one done by a professional may well be a very good step to take. However, in order to improve the contribution of testing to product quality, you need to understand what can realistically be expected from testing in the whole area of all your processes. Remember that from a CMMi®[4] point of view, you need a fairly high maturity level before testing is really properly addressed. So some relevant factors for success:

1. Even if you have a good test policy, to what extent can you influence the development organization with it? How can you deploy it? How do you enforce adherence? Would you want to use enforcement? Why? Why not?
2. What test metrics can you really get from the organization? On whom may you have to depend for tooling? Is there an internal IT department? Which other branches of the company do you have to cooperate with? How do you train defect analysts? (Who will fulfil that role?) How do you keep them motivated to fill in the right information, even after months? What would you have to measure in order to find out which improvements contribute to the business and which ones do not?
3. If you start test automation or unit testing, how can you relate the value of the defects found by these tests to the value you add because otherwise they would have been found later?
4. Does your organization have the skills it requires?
5. Do you actually know if tools for testing are already in place?
   a) What state are they in?
   b) How much legacy testware and how many assets do you already have?
   c) How do people feel about these tools?
6. Can you actually start to test? That is: Is your requirement engineering good enough to allow for testing? One practice I had to do is actually to write down the assumed requirements as part of a test design. Far from elegant to be sure, but better than nothing and enough to make people ashamed enough to actually start writing requirements.

From my experience, I would say it is very good to start off with a fairly clear idea about these questions before starting to address the pain points. At the same time, you have to start somewhere, so start with a pilot and see what you learn along the way. Do not expect instant excellent results. Be prepared to encounter all sorts of obstacles no one had even thought existed.

How to involve your stakeholders

Many of your stakeholders (do you know who they are?) may not know much about testing. They may severely overrate their own knowledge and underestimate how difficult it is to get a good test

2 *Failure Mode Effect Analysis*

3 *Food and Drug Administration*

4 *Capability Maturity Model (integration)*

process in place.[5]

Depending on the influence they have, you will have to adapt the services you can offer to the needs they have, and always address these needs. Tongue-in-cheek, there is the ‚Engelsma effect' that states that once you have simplified the issue so much that a manager understands it, he thinks it actually IS simple, and you do not get the resources you need...

It works best by finding small but tangible advantages for your stakeholders. Find early adopters and with them show first results. At the same time have management back up your plans (Have you got a roadmap of how you want to get to your vision? What is your vision, stated in language the others understand? Which steps must you logically take?), not just by resource allocation, but by the direction they provide and the control they exercise.

If you really cannot find enough management support, you know the time simply is not yet ripe for your ideas. Do not get frustrated, but accept the reality.

Limiting your scope

One situation when you start to improve the test process you may likely encounter is that you find you cannot improve much unless other processes that are outside testing are improved. I have known one situation where the test manager got the blame for the cost of testing being too high. The root cause of the testing time was that SW installation could hardly be carried out due to bad SW quality. And the manager had not clearly indicated the limits of his influence and responsibility!

So clearly find the facts, but avoid the blame game.

In your test policy identify what assumptions you make about the environment you work in, and what you expect from others. Here, too, entrance criteria DO apply!

Track and celebrate

So, you have your vision, you have set out the test policy, you have identified the small steps that you can set, you have got the go-ahead from your management, you have found the early adopters. Now make sure that you track all those little steps, make sure you log their results, and share these. One thing that keeps on surprising me is how good real testers are at not even noticing their achievements, let alone sharing them! And whenever you reach a real milestone, celebrate, show to the larger development population what it means to them. And have some fun with the guys who helped you make it come true.

> biography

Erwin Engelsma
is a SW Test Architect with Philips Healthcare. He has also had almost 10 years of experience as head of the department for testing and system engineering. He has set out roadmaps for streamlining and introducing testing tools in complex organizations, and is a driving force in improving software test processes. He also drives SW reliability process improvement. One of his biggest motivators was (at the very beginning of his career) his involvement in a project automating a robot in a bread baking factory. The project did not succeed, and he has ever since been wondering what really went wrong and how you can learn from that experience. His contribution to a software teaching book about integration in platform based system development is a partial answer. When not doing technical stuff, he integrates loads of notes into a music band that performs regularly at weddings and cultural festivals.

5 *This is known as the Dunning-Kruger effect, a cognitive bias in which unskilled people make poor decisions and reach erroneous conclusions, but their incompetence denies them the metacognitive ability to appreciate their mistakes. They therefore overestimate their abilities. Conversely, people who have more knowledge often underestimate their abilities and are less self-confident. Hence unsuspecting managers who can only act on shows of confidence pick the wrong guy.*

HANDBUCH

TESTEN
IN DER FINANZWELT

Das Qualitätsmanagement und die Software-Qualitätssicherung nehmen in Projekten der Finanzwelt einen sehr hohen Stellenwert ein, insbesondere vor dem Hintergrund der Komplexität der Produkte und Märkte, der regulatorischen Anforderungen, sowie daraus resultierender anspruchsvoller, vernetzter Prozesse und Systeme. Das vorliegende QS-Handbuch zum Testen in der Finanzwelt soll

- *Testmanagern, Testanalysten und Testern sowie*
- *Projektmanagern, Qualitätsmanagern und IT-Managern*

einen grundlegenden Einblick in die Software-Qualitätssicherung (Methoden & Verfahren) sowie entsprechende Literaturverweise bieten aber auch eine „Anleithilfe" für die konkrete Umsetzung in der Finanzwelt sein. Dabei ist es unabhängig davon, ob der Leser aus dem Fachbereich oder aus der IT-Abteilung stammt. Dies geschieht vor allem mit Praxisbezug in den Ausführungen, der auf jahrelangen Erfahrungen des Autorenteams in der Finanzbranche beruht. Mit dem QSHandbuch sollen insbesondere folgende Ziele erreicht werden:

1. *Sensibilisierung für den ganzheitlichen Software- Qualitätssicherungsansatz*
2. *Vermittlung der Grundlagen und Methoden des Testens sowie deren Quellen unter Würdigung der besonderen Anforderungen in Kreditinstituten im Rahmen des Selbststudiums*
3. *Bereitstellung von Vorbereitungsinformationen für das Training „Testing for Finance!"*
4. *Angebot der Wissensvertiefung anhand von Fallstudien*
5. *Einblick in spezielle Testverfahren und benachbarte Themen des Qualitätsmanagements*

Herausgegeben von Norbert Bochynek und José M. Díaz Delgado

Die Autoren
Björn Lemke, Heiko Köppen, Jenny Siotka, Jobst Regul, Lisa Crispin, Lucia Garrido, Manu Cohen-Yashar, Mieke Gevers, Oliver Rupnow, Vipul Kocher

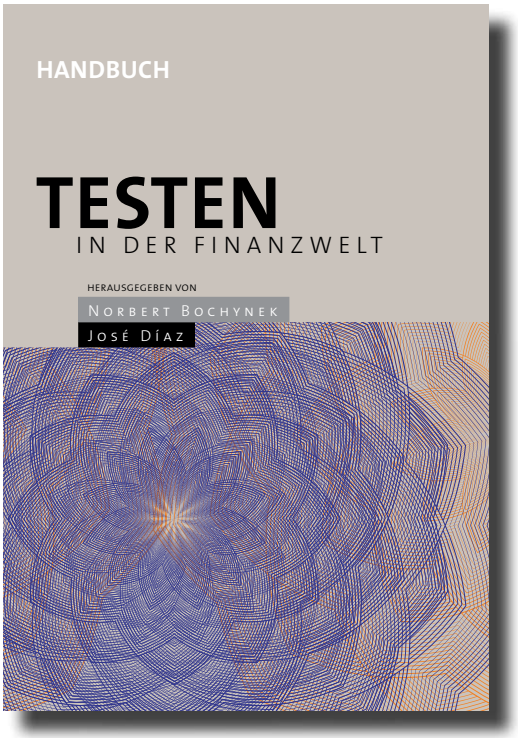
Gebundene Ausgabe: 431 Seiten

ISBN 978-3-00-028082-5

1. Auflage 2010 (Größe: 24 x 16,5 x 2,3 cm)

48,00 € (inkl. Mwst.)

www.diazhilterscheid.de



Automating Innovation

The rise and impact of automated testing tools and technology

by Christopher Eyhorn

The year was 1962 and the world was catching its first glimpse of the daring and colorful wonders that tomorrow might bring. It was a space-age vision of a technology-filled future – flying cars that could fold down into an attaché case, wisecracking robotic maids, and miraculous machines that could do anything from washing the dog to cooking a three-course gourmet meal, all at the touch of a button. It was a utopian society made cleaner, better, and more prosperous through the auspicious use of technology and automation.

This description, of course, refers to The Jetsons, that iconic animated family, with their pneumatic people-mover tubes and moving sidewalks. Technology and automation were a key factor in improving the lives of the denizens of Orbit City, where George Jetson lived and worked as a full-time engineer...full-time being a whopping nine hours per week, thanks to a multitude of labor- and time-saving devices that were an everyday part of life for George and his family. Although our modern society hasn't reached the same utopian apex as Orbit City, there are disruptive shifts in technology automation that have changed the face of our world today and continue to have an indelible impact on businesses and consumers alike.

Take for example, the software development landscape. Facing a challenging – and frequently unruly – economic climate, individual software developers and project teams are often tasked with doing more yet having less to do it with. In Agile environments, that need is even more pressing; minimizing waste and risk, while maximizing productivity and velocity are essential for sustained success. How to achieve better results with fewer resources might seem an impossible dilemma; however, for those willing to take a chance on today's innovative, emerging technologies, there are indeed solutions that make sound technical and financial sense. And one such solution is that of automated testing tools.

Putting Agility to the Test

B. Bereza-Jarocinski of the Global Association for Software Quality once said that "introducing test automation is sometimes like a romance: stormy, emotional, and resulting in either a spectacular flop or a spectacular success." The metaphor is a fitting one – it is true that interleaving automated tools within the test environment can be a tumultuous process. Introducing test automation is an event that can bring with it tangible benefits and the opportunity for achieving greater success. But it can also result in abject failure, thwarting project goals, frustrating teams, and dragging down ROI.

Scoring a favorable outcome when adding automated technologies to any testing environment depends on untold numbers of variables – testers' willingness to try and acceptance of test automation solutions, team composition, organizational priorities, budget and schedule constraints, and the quality of the tools being introduced. However, since their debut in the 1980s, automated testing tools have evolved rapidly, migrating swiftly from their core functional testing capabilities into sleek, sophisticated, and intelligent integrated suites that raise testing accuracy and team productivity. Today's cutting-edge test automation technologies have resulted in a generation of robust products that make automating tests faster, easier, and more cost-effective than at any time ever before.

With the ubiquitous adoption of Agile development methodologies, any enabling technology that permits teams to produce high-quality deliverables on-time and on-budget is a valuable resource. In the results-oriented Agile environment, the benefits that test automation offers cannot be understated:

- **Accelerated development cycles** – Short sprints require features to be fully developed, tested, and debugged in mere weeks, if not days. With this increased emphasis on Agile, the need for automated testing has risen, as well. Unlike manual practices that can result in lengthy, drawn-out test cycles, automated testing approaches allow scenarios to be performed swiftly and repeatedly without the need for manual intervention. By optimizing test processes for the utmost efficiency and speed, teams can successfully achieve and sustain durable velocity levels. Lastly, test automation introduces greater scalability into the test environment. In allowing tests to be run simultaneously rather than serially across any number of machines within the testing environment, automation shortens total execution times.

- **Improved resource utilization and productivity** – In contrast to earlier generations of testing tools that were cumber-

some, overly complex, and had steep learning curves, current test automation technologies are an evolutionary leap forward. Intuitive, intelligent, and easy-to-use, today's automated testing toolsets improve the allocation and use of limited project resources. For example, with a wider array of options, functionality, and capabilities, innovations within test automation are enabling non-technical personnel to function effectively within the test environment. In turn, developers can be relieved of some testing responsibilities, permitting them to instead focus on other critical work at hand, elevating overall project productivity.

- **Enhanced team dynamics** – As McDonald's founder Ray Kroc once noted, "None of us is as good as all of us." Collaboration is a keystone of Agile; without a tightly integrated team pulling together towards a common goal, velocity decelerates, impeding success. Agile teams often employ conventional testing practices, which can yield less than optimum results. Relying on such traditional methods can silo developers and QA personnel by putting them at odds, stifling collaboration, and blinkering project velocity. Introducing automation allows teams to rethink their testing approach and reset perceptions of how and where testing fits into the development process. By leveraging automated testing tools, silo walls can be removed, improving team collaboration and dynamics. The outcome is high-powered performance delivered by teams better functioning as cohesive, tightly integrated, organic units.

With Agile as the new reality for many developers, project teams, and organizations, finding conduits to greater productivity and sustainable velocity is mission-critical. By availing themselves of the out-of-the-box benefits and value that test automation offers, testers and developers alike will be well prepared to meet the challenge of achieving their objectives faster, easier, and with fewer resources.

A Look Behind the Curtain

Having examined the benefits that test automation offers, it might be helpful to take an in-depth look at how test practitioners can put automated testing tools to use in real-world situations. No two teams are ever exactly alike – what works for one may spell trouble for another. However, with a better understanding of test automation mechanics, it becomes possible to see how it can successfully be applied in any development environment.

Properly identifying a team's goals for using automation is an important first step when starting a new project and is especially important for teams resurrecting their automation efforts for existing projects.

General considerations teams should remember to take into account:

- Get started with the automation effort as early as possible.
- Remember that it is critically important to have clear direction from leadership about how and what to automate.
- Don't fall into the tempting trap of trying to write the automation framework internally, as this attempt will become just another development project competing for limited resources.
- Automate end-to-end scenarios and not just middle level scenarios.
- A test is never finished until it successfully runs in the execution environment.
- At the end of the day, automation technology can only take a team or a project so far; the overall attitude toward quality is equally important.

Key qualities to look for in a good automation tool:

- It should help everyone take ownership of quality by serving multiple roles, and allowing all stakeholders for the project to construct tests and understand their purpose.
- The ability to support a test-first approach by using a hybrid of automated and manual steps. Teams should leverage the power of automation but with manual steps for user interface that isn't finished or is changing. Doing this allows teams to build more automation faster.
- Easy integration with the same source control system that the project code base uses. This eliminates the barriers that fragmented systems create when trying to debug tests and resolve bugs in code.
- Facilitation and fostering of greater collaboration – collaboration is of the upmost importance for successful automation efforts on any Agile project.

What to automate and things to remember when introducing automation:

- User stories and dependency-free requirements. Start with user stories or requirements that do not have dependencies that can break automated tests. Before automating a scenario, teams should always analyze areas of the project that are interconnected and still under development in order to minimize the fragility of a given test.
- Important business scenarios. Always focus on testing those scenarios that make the tool valuable to the business. For example, if the organization is an airline, the first test the team should consider automating is ticket booking.
- Complexity vs. regression value. The desire for a manual tester to find a bug often leads them to try the most obscure, random, and crazy scenarios possible. In automated testing, this is suboptimal – once this obscure bug is found, an automated test for that scenario is essentially worthless and has zero regression value. Very complex tests should always be weighed against the regression value they will bring to the project.
- Write the happy path test first and then expand to complex or edge scenarios.

"Testing is Human, Quality is Divine"

While this saying might seem to run counter to the concept of automating testing, the opposite is actually true – any tool is only as good as the person using it. Stated plainly by Aristotle, "Quality is not an act, it is a habit." Applied to the testing arena, this concept means that no matter how automated the environment, the human touch will always be essential to achieving meaningful, measureable success. And indeed, that human element might be the most important one of all.

Once the sole domain of software developers and those experienced testers with the technical proficiency for building and executing test scenarios, the ongoing evolution and maturation of automated testing tools is reshaping the erstwhile software testing paradigm. With new test automation tools, less skilled and even non-technical test personnel are realizing productivity levels rivaling those of traditional professional testers. The suggestion that non-technical personnel can be a valuable addition to the testing discipline will likely bring cries of protest from those

testing experience

Díaz Hilterscheid

Already Certified?

Join the Alumni Scheme and keep your knowledge up to date

Aimed at individuals and business, the Alumni Scheme provides those already certified the opportunity to gain ongoing access to latest versions of course materials plus many other benefits for a nominal annual subscription.

- 50 e-books on Testing and related IT
- Latest version of certificated courses
- Regular testing content updates
- Discounts on Learntesting products
- 12 months 24×7 access
- For Individuals and Business
- The Learntesting Alumni Scheme

Supporting professionalization of the Software Testing Industry

The Learntesting Alumni scheme – lifelong learning and support for Continual Professional Development (CPD) At Learntesting, we take seriously the continual professional development of software testers. After all, if we want to be recognised as a profession, we should start behaving like one! In October 2010, the new release of the ISTQB Certified Tester Foundation Syllabus introduced a 'Code of Ethics' which includes:

"Software Testers shall participate in lifelong learning regarding the practice of their Profession"

There is no concept of 're-certification' in the ISTQB scheme at Foundation or Advanced Level.

The fact that 150,000 individuals have been certified against many different versions of the syllabi over the past 13 years, means that one of the fundamental principles the scheme supports, 'A common language for testing', is compromised. Anyone certifying against the current Foundation syllabus can now be speaking a very different language to someone from several years ago – even if they could all remember it all!

At Learntesting, we have devised an innovative scheme that supports the new ISTQB 'Code of Ethics' and the concept of CPD for Testers. The scheme allows anyone already certified to have ongoing access to the very latest materials plus a whole range of other benefits – all for a nominal sum.

www.learntesting.com

testers who have invested years honing their craft, yet it is an idea that should not be rejected out of hand. It should also be noted that having non-technical testers is not a novel concept, as they are commonly integral members of manual testing teams. What is new however, is the idea that they can become vital players in automated testing settings.

With the rise of intelligent tooling backed by intuitive, point-and-click interfaces, the learning curve for automated testing tools is negligible. By eliminating barriers to entry and adding simpler-to-use elements such as conditional logic support, these sleek, easier-to-use, and more powerful tools allow non-technical testers to be swiftly added to the test mix. They can now quickly learn how to efficiently and successfully write reusable automated tests without the assistance of developers or technical testers. This provides a richer project resource pool that teams can call upon to attain better overall productivity, accelerate the testing cycle, and realize greater added value and ROI.

There are other benefits that are less obvious, as well, and these can have a profound impact within Agile environments. For example, they bring with them a broader perspective uncolored by previous assumptions – they may try to use the test subject in ways developers had never intended, uncovering hidden bugs and enabling them to capture scenarios that might not otherwise have been tried. Furthermore, this lack of supposition can also provide insight into the end-user mindset, allowing for better refinement of features, functionality, and UI.

While the benefits of non-technical testers are manifold, there are considerations that must be taken into account. Before dashing over, grabbing Sally from Accounting, and ushering her into the test lab, teams should consider what traits a good non-technical tester should possess. Non-technical testers should be curious, willing to ask questions that others (like business analysts) might not, and have a desire to learn more than just the functional aspects of testing. They must be willing to take on a role as a cheerleader for product quality and not be easily discouraged. In turn, teams must rethink their opinion and treatment of non-technical testers – they should be seen as valued team members, rather than adversaries or interlopers because despite the continued advance of automated testing tools, all testers will continue to be a lynchpin for success in the software development landscape.

Improving the World One Bug at a Time

Test automation has come far since its first appearance in the software development landscape. New automated tools are elevating the profile of the testing discipline, giving it added relevance, importance, and value, particularly in Agile environments. While automated testing tools may not be as impressive as creating the world's first flying-suitcase car or usher in the age of the nine-hour workweek, they are truly essential in providing better, faster, richer, and more rewarding experiences for testers, developers, and ultimately, the end-users they serve. In the end, the truth of the matter is that test automation tools are the channel that will allow developers and teams to improve the world...one bug at a time.

> biography

Christopher Eyhorn
is the Executive VP of Telerik's testing tools division where he is building the next generation of automated testing tools. Formally co-founder and CEO of ArtOfTest, he has written automation frameworks and functional testing tools and has worked in a variety roles ranging from developer to CEO within the company. Christopher has worked with a variety of companies ranging in size and industry. He is a licensed pilot that loves to fly every chance he gets and truly appreciates the importance of hardware and software testing every time he takes off.

End2End Testing: It's Just the Beginning

by William Gens

Having spent years working on „new" development projects, projects often started from `scratch, I never much encountered the concept „End2End" testing until recently. The systems I worked on were still in the early phases of parallel production, and so many changes were going into the system, not unlike transients passing through those wild old west mining towns. We architected our testing to cover everything and test almost everything in the shortest amount of time.

As systems settle down and the wild old west begins to resemble a nice suburban community, so too does the testing. It is at this point that the group's testing lead begins to propose to testing and development management a more efficient and less resource intensive way to test. This production or post development phase, or what is often called the maintenance phase, isn't something I was too accustomed to in my career.

In the last few projects I've been on, the testing methodology never changed or transitioned from development phase to post-production or maintenance phase. The scripts remained the same and the size of the test suite was equivalent to using a sledge hammer to drive a nail.

It doesn't take the seasoned QA professional long to quickly recognize the red flags of post production development, maintenance and testing. Most recently, I was looking at an investment banking application which took 85 man days to test. I had to ask a few times were those hours or man days? Man days I was told. I immediately thought of some large projects I was the QA lead/manager on; 30 million type development projects, and thought, I had to test a large scale application across multi platforms with a complex compliance and trading function in 1 week or less! How could this be, I ventured to ask? No one seemed to know the answer, but a few ventured to say they didn't really think about it; they had gone through the testing for so long they only used them for indicating test results.

The testing manager poured over these voluminous testing tomes to indicate for each release what was in scope and what wasn't. These are scripts all defined and organized in Excel spreadsheets. This is all going on while they are in the midst of automating through QTP a lot of the applications functionality.

I was asked to look at this testing process and make some recommendations. It didn't take a testing expert to figure out that so much of their testing effort was all effort without the expected results and return on the testing team's investment.

Here is where a simple solution comes in. Identify 10 critical transactions end2end with the application. Usually these types of critical transactions touch upon the breadth of the system. Depth may or may not be necessary at this phase of the application's life. What I had found in those thousands of test scripts were countless tests against display, navigation and configuration. While this may have been necessary a liftetime ago with this application, there aren't many changes that would impact display and data driven testing. Of course, this doesn't hold true if the application is rewritten or refactored, or if WEB based and it now must run on different Browsers or release of IE, which directly impact display and navigation.

For the most part, those „10" critical transactions that go across most functional points of the application should provide a very high degree of coverage. After all, testing business logic and transaction flow is the most critical area to test.

As you begin to build upon those workflow tests with ad-hoc regression tests, your workflow tests become like the apple tree and the regression tests like the fruit that hangs from the tree. Depending on what changes are made to the application determines how you enhance those workflow tests with more detailed tests around the central workflow ones.

Once you've done this, you can begin considering automation. You now have a tightly well defined set of scripts and these workflow scripts are the ones you address first and foremost in automation. It is a common and costly mistake to automate scripts that were poorly architected in the manual testing phase. Most automation specialists are more technicians than QA architects or analysts and usually just start working towards automating input and output results and building an automated suite that

mirrors that of manual testing which is very resource and time intensive.

Don't be afraid to take those legacy scripts and throw them away and re-achitect your testing. Identifying workflows and testing towards that is fun for the testers and eliminates the drone of testing hundreds of useless scripts. And remember, the scripts you write aren't meant for monkies or the proverbial person off the street, they are meant for testers with varying degrees of familiarity and expertise in testing an application.

Finally, tracking these tests and reporting on them, preferably in a test management tool, is easy and meaningful. If you identify the workflows by component and sub-component within the application, your test management will reflect that as well. No one is interested in tracking issues or test results at a granular level; they are, however, interested in tracking the results and coverage at a component or sub-component level.

The benefits to this approach in manual testing and transitioning to automated testing will pay huge dividends in the efficiency of testing and the resources allocated. Your testing group will be excited to take on new projects and learn new things instead of being mired in voluminous unnecessary testing and overhead. End2End testing - it's the beginning of how you architect or re-architect your testing.

> biography

William Gens
a long-time New York resident, has been in Quality Assurance in the Financial Services industry for over 25 years. He has held many positions at such prestigious firms like Bear Stearns, ADP, Credit-Suisse First Boston, CreditEx and Dealogic. He also spent many years as a private consultant in various firms. He is an expert in offshore QA management. When not testing, you can find him coaching and playing squash near his Long Island home or writing about squash on his popular blog: squashdashersbashers.blogspot.com.

Introducing a Model-based Automated Test Script Generator

by Martin Steinegger & Hannu-Daniel Goiss

This article provides an introduction to a new way of generating test scripts for GUI testing (web based or rich clients) using a model-based testing approach and describes how this alters the test process. The described test code generation tool (Abstract Testing Tool) was developed by the authors in cooperation with one of our clients. The generated automation code can then be used for performance testing or functional testing.

The abstraction layer, which is at the core of this approach, will be explained at the beginning. Afterwards we will give an overview of how this approach changes the testing process and discuss advantages and disadvantages.

Introduction

The most common way to create test scripts for test tools is (unfortunately) still „capture and replay", even though the benefits of treating test automation like a software development project have been demonstrated repeatedly. In a capture and replay approach, the test tool simply records the users' actions and generates a script from them, which can be used to execute tests. This approach has several drawbacks. Every step has to be captured each time a script is generated, even if the step is contained in every script. Although tools employing a modular approach have been developed, they still require in-depth knowledge of the testing tool and scripting language, as it is necessary to adapt the test scripts after capturing them from complex applications in order to make the scripts more robust.

These problems can be addressed by using an abstraction layer, which we will call the "Abstract Testing Tool (ATT)". ATT introduces an abstraction layer, which separates the testing tool from the test script and introduces a representation of the System under Test (SUT) with all its pages and navigation paths and a meta-scripting language. The tester generates scripts by navigating through the SUT in a GUI, provided by the ATT.

Abstraction Layer

This section gives a short overview of how to create the tool's abstraction layer. The following key topics will be explained: **test object model, test script and code generation**.

The test object model contains a representation of the SUT with all its pages and navigation paths. The test scripts are scripts, which test the application by navigating using the navigation paths that have been stored in the test object model. Code generation is the process of creating scripts for the test tools, combining the test scripts and the test object model.

Test object model

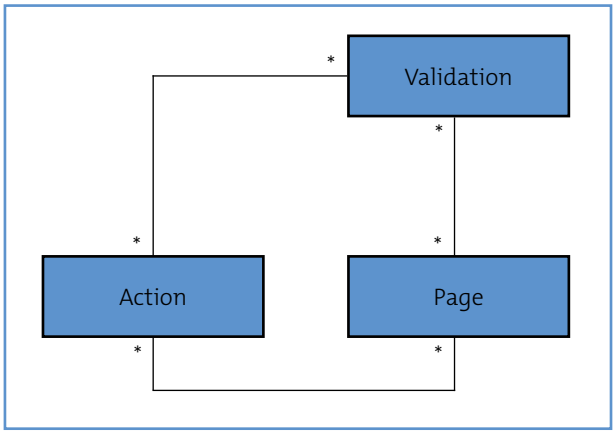


Figure 1: Test object model

The test object model is an abstract model representing the test object. The minimum requirement is that it contains the possible interactions within the SUT (actions), which will form the test scripts. For example, on a website these interactions would be clicks on links or buttons. The action in the object model has to contain every component (for example parameters of a link) necessary to be performed.

Pages and a mechanism for verification should also be part of the test object model. Pages represent start and end points for actions. For example, google.com and the search results could be pages, while the search button (with all the necessary parameters) would be an action, leading from the start page to the search result page. Verification objects are a mechanism to check the accuracy of the end points of actions. Usually verification objects are a set of strings, pictures or other elements that are part of the end point. For example, for a Google search action, the expected result list could be a verification string. If the end point of the action doesn't contain the search action, the verification will not be successful. Verification objects can be connected either to actions or pages, and there are many different possible types, like

positive or negative checks, etc. The model can thus be arbitrarily complex. For example, events can be added to verifications to react on errors. Possible reactions could be error logging, creation of screenshots or environment snapshots.

The test script

The test script is basically a list of actions that have been defined in the test object. The meta-scripting language itself can be created in many different ways (Domain Specific Language), but it should be able to be read by a human (not that developers are not humans, but we digress...).

The tester should be able to create test scripts using a GUI. Using UML activity diagrams for representing test scripts in the GUI could be a fast way to create such a GUI, but can lead to accuracy issues because the UML editors are not made for script creation. An own language will bring more flexibility and accuracy. But activity diagrams can also be used for visualization of your own language.

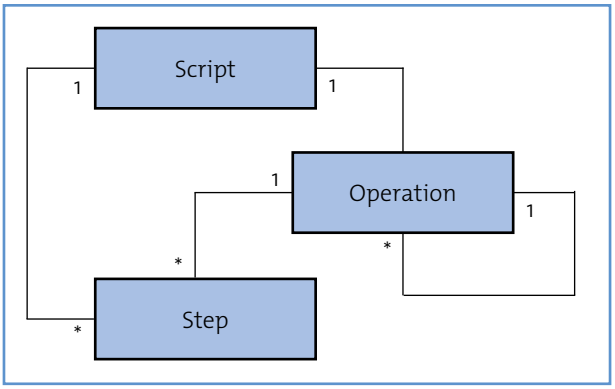


Figure 2: The test script

The following is an example of how the scripting language could look. The object "script" is the root node and contains many steps. An example in XML could be:

```
<script>
<step link="technical or functional key from the
test object model"/>
<operation/>
<step link="technical or functional key from the
test object model"/>
</script>
```

A step is an action the user can perform within the application. Every step needs a reference to an action from the test object model. It is important to distinguish between steps and actions, because actions are unique (for example there is only one action for performing a Google search), but the same action can be referenced to many steps in a script (the same Google search can be performed multiple times in the same script).

Operations represent control structures that can affect the script flow or commands that have to be executed. Complex scripts have to be nested (e.g. if – else, loop exit, log, delay, run, waitUntil, skip, store, check).

Functional scripts are often flat, because they have the focus to check exactly one piece of functionality at a time. In contrast technical scripts are rather complex due to contingencies that appear because of the test data and necessary flow controls.

```
<operation type="definition" >
<step link="..." />
<step link="..." />
<operation type="..." />
</operation>
```

When deciding which operations to implement, the targeted test tools have to be taken into account. Some operations might not be possible with all test tools, so they may require add-ons to the test tool. This is important as this approach can ensure test tool independence if the scripts are to be used with many different test tools.

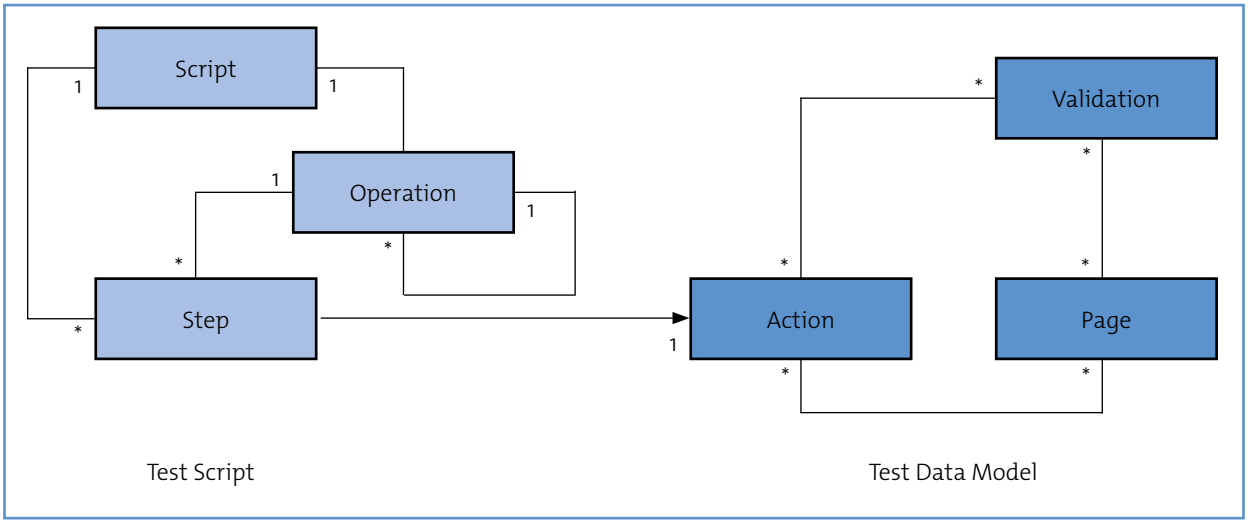


Figure 3: Combining the test object model with the test script

Abstraction layer for the code generation

This section describes how the test object model and the test script are being transformed to source code (for the testing tools) or documents. A generator engine is needed to merge the test object model and test script. The output of the generator is an object graph, which will be used as input to create test source code itself. This mechanism is very useful to change content before it will be sent to a template engine (templates will be described in the next paragraph). Figure 4 is a visualization of a merged test object model and test script.

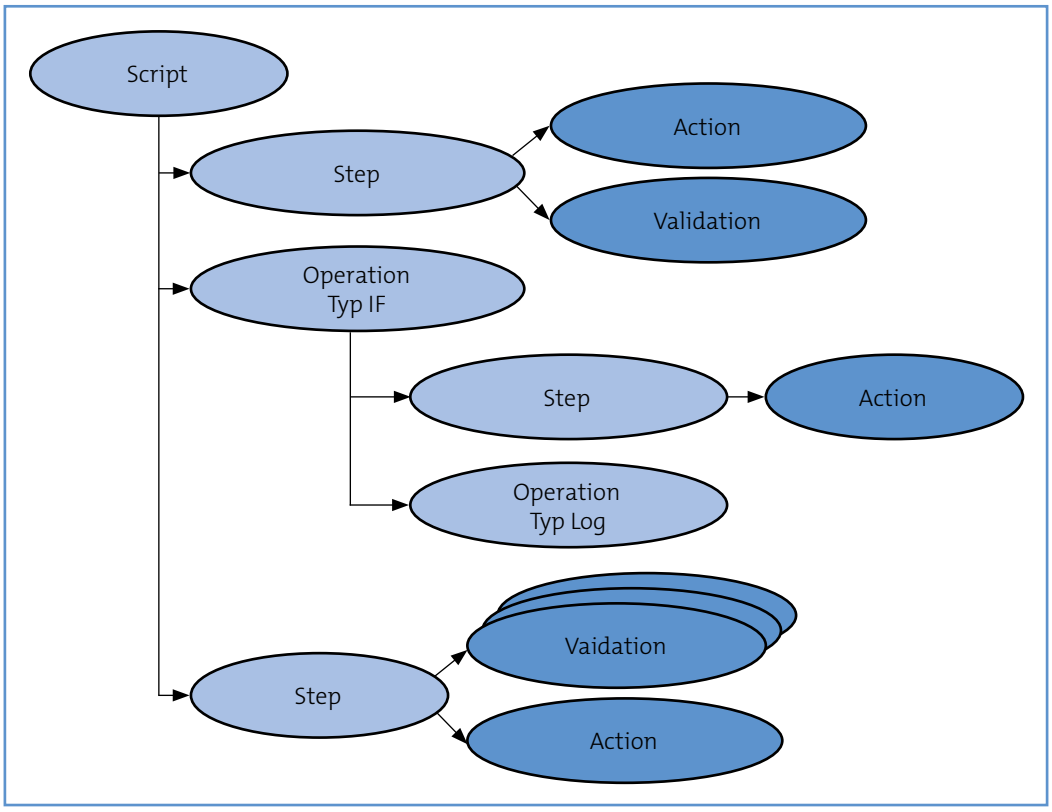


Figure 4: Merging the test object model and the test script

Template engines (e.g. Velocity, Xpand or Hamlets for Java) can be used to generate code. This technology has the advantage of being very flexible. The engine can be used to convert a graph to the desired format. Templates have to be developed for every desired format. For example, one template could be created to generate HP LoadRunner code, and another could be generated to create Micro Focus SilkPerformer code. However, template languages can be tempted to fudge (duplicate code, ...), therefore the template code should be kept structured and modular. Template languages are not made for complex logic. If it is necessary to create complex logic, it can be delegated to the target programming languages (for example C oder Java) by creating a function call.

Keep the following points in mind:

- Don't repeat yourself

  The most important thing is to create a test object model that is maintainable by avoiding redundancy.

  E.g. to reduce duplicate interactions that exist on many pages (e.g. main navigation), they have to be defined one time and, if needed, referenced.

- Keep it simple

  Think about the complexity and your requirements. The scripting language should be as easy as possible. Only implement features that are really needed and cannot be replaced by a combination of simple features. Complex functions are rarely used and not required.

- Don´t forget versioning

  Changes to the model and the scripting language can occur with every release or deployment of the SUT. Many people have to be able to create scripts and make changes in the model. This is a frequent source of errors. These changes have to be traceable and enable rollback. Versioning is a key feature.

How does the tool change and improve the test process?

The graphic illustrates the difference between conventional test script creation and test script generation using ATT. The main difference is that in the conventional way the main task is to use the capture functionality to generate the script and to make manual adaptations until the script is capable of running. Using this approach, the most cost-and time-intensive part is manually adapting test scripts (and often multiple scripts for the same reason), as in large applications the test script usually has to deal with a wide variety of pages and test data.

Using the ATT approach, the most cost- and time-intensive task is filling and maintaining the test object model. Once the test object model is filled with the bulk of the SUT, almost all test scripts can be generated quickly.

In the conventional approach, the most cost-intensive task (manually adapting the code of the test scripts) has to be performed every time a script is generated, while in the ATT approach the most cost-intensive task (filling the test object model) has to be performed once, as during the test phase the test object model only has to be maintained in case of changes in the application.

The following are some **advantages** of ATT:

- Test scripts are generated independently from the test tool in use
  - It is possible to use the same test scripts in different test stages (for example test scripts can be generated for different performance test tools for the performance testing stage and various scripts for automated test tools can be generated for the functional testing stage).

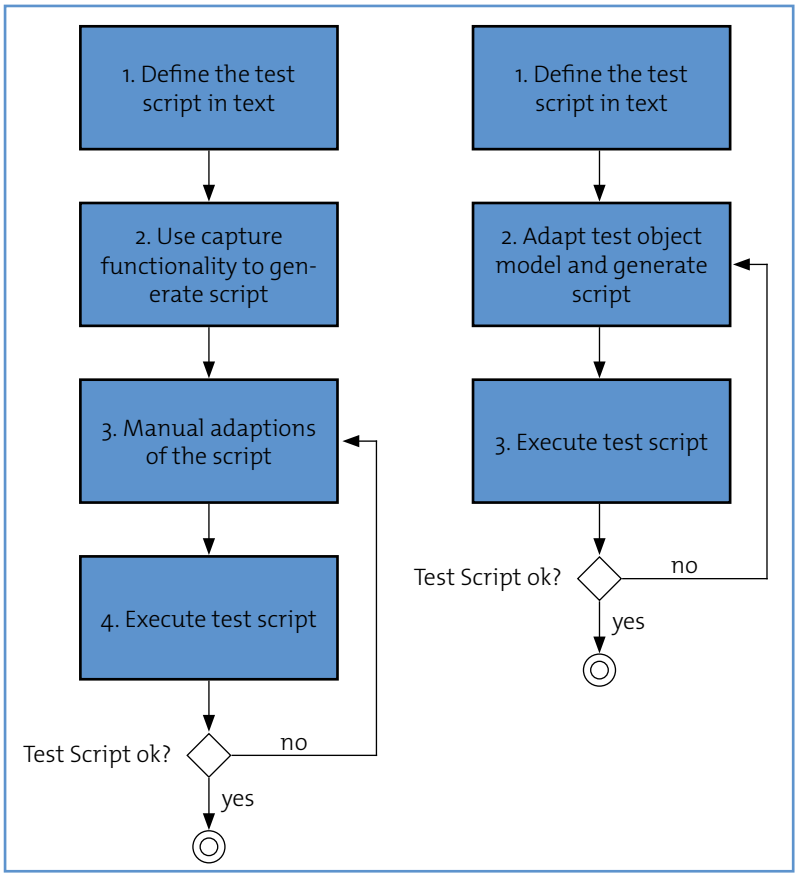


Figure 5: Changing the test process (left side: conventional flow; right side: test script generation using ATT)

-> This changes the test process significantly. Usually, one team is responsible for generating and executing test scripts for each test phase. With the ATT approach, one team can be in charge for generating test scripts for the whole test process, while the test stage teams only need to be responsible for test script execution and analysis.

  - In case of changing the test tool (for example a switch from one vendor to another) all test scripts can be reused. The use of a new test tool requires the development of a new template and the regeneration of the test scripts using this new template.

Regression tests can be defined independently from the test tool and can be used with any test tool.

- Changes in the SUT do not require direct changes to the ATT test scripts.

Should changes be necessary, the test object model has to be adapted and test scripts can be regenerated. This significantly reduces the maintenance effort, as changes only have to be done at one central location (test object model).

- Test scripts can be generated without specific knowledge of the test tool and of the programming language used.

Manual adaptations to test scripts, which require programming skills, are usually not necessary. As discussed earlier, ATT and its functionalities should be kept simple, as complex functionalities make it more difficult to use. This makes manual adaptations necessary in case of functionalities that have not been implemented in ATT. For example, if ATT does not have the possibility to test AJAX functionality, the generated scripts have to be adapted manually.

  - This makes it possible for "functional" test team members to generate test scripts, as it does not require programming skills. The only thing they have to do is to navigate through the test object model to generate their test scripts.
  - A special training period is not necessary when switching the test tools because the test scripts are automatically generated by ATT. It is still necessary to train the responsible persons to execute the test scripts and analyze test results.

- It makes sense to use ATT if there are steps in the test scripts that can be reused in other test scripts. The introduction of ATT does not make sense if every script is unique.

Some of the disadvantages of ATT are:

- The test object model is a frequent source of errors. Granting write access to a high number of people should be avoided. They might change things that are still valid and should not be changed because of impacts to other test scripts.

  -> Only a few people with in-depth knowledge of the test object model should be allowed to make changes. Change control and review processes for each version are important as for any other test artifact.

- As ATT should be kept simple, some parts of the test object might not be testable using only ATT. For example, when testing AJAX in a web application, it will not be possible to test this if ATT does not provide the possibility to do so. The require-

ments for ATT need to be continuously validated and updated in the ATT where necessary.

- While ATT adds test tool independence to the testing process, it generates a new dependence on ATT.

Conclusion

The described approach adds a new test script generation approach to the testing process. It has been successfully used in practice and has been proven to be effective with regards to:

- time and cost-efficiency
- test script generation for different test tools
- re-usage of functionality across different test scripts
- usage of test scripts with a SUT, which is still under development and has frequent GUI changes.

> biography

Martin Steinegger *works at Accenture in performance and security testing, while* **Hannu-Daniel Goiss** *is involved in performance and integration testing. They have both been with Accenture for over 2.5 years, working for multi-national clients in various industries. They hold certifications such as ITIL Foundation Level and ISTQB® Certified Tester Foundation Level.*

Subscribe for the printed issue!

Please fax this form to +49 (0)30 74 76 28 99, send an e-mail to info@testingexperience.com or subscribe at www.testingexperience-shop.com:

Billing Adress

Company: _____

VAT ID: _____

First Name: _____

Last Name: _____

Street: _____

Post Code: _____

City, State: _____

Country: _____

Phone/Fax: _____

E-mail: _____

Delivery Address (if differs from the one above)

Company: _____

First Name: _____

Last Name: _____

Street: _____

Post Code: _____

City, State: _____

Country: _____

Phone/Fax: _____

E-mail: _____

Remarks: _____

1 year subscription

☐ **32,- €**
(plus VAT & shipping)

2 years subscription

☐ **60,- €**
(plus VAT & shipping)

Date

Signature, Company Stamp

Domain based testing is here!

– The next wave of QA innovations

by Shishank Gupta & Rajneesh Malviya

QA practices over the decades have seen many waves of innovation, be it test automation or techniques to optimize testing. In the early part of the decade, if one spoke of matured QA practices, it referred for the most part to standardization of processes, common metrics for measuring QA effectiveness and, probably, test automation. With changing times, the QA function as we knew it evolved and the focus shifted to:

- Requirements coverage vis-à-vis code coverage
- Meeting relevant compliance and regulatory standards
- Keeping in mind the end user perspective, while certifying the application under test

A common thread to the aforementioned aspects is the need for QA teams to have a deeper appreciation of the business domain of the application being tested. This trend is one of the most pervasive phenomena the QA industry has seen lately. The software QA industry, which had a largely horizontal focus and was indifferent to business domains, is now turning into an industry where specialization in verticals is a must.

For QA teams to be seen as true advisors to the IT leadership, their ability to understand the domain in which the customer is operating and take decisions that are in line with the customer's business priorities is critical. Be it to prioritization of requirements based on business criticality or reporting business impact of unresolved defects, the QA teams have a significant role in ensuring that IT applications help customers derive maximum business benefits from their investments in IT.

Domain specialization helps QA teams achieve multiple benefits through the Software Development Lifecycle (SDLC). The table below summarizes the potential benefits of a vertical focus in QA through the SDLC lifecycle.

Life cycle stage	Benefits of vertical specialization
Requirements analysis	• Ensure coverage from business functionality perspective • Enables early identification of defects through static testing
Test planning	• Prioritization based on business criticality
Test execution	• Risk based test execution • Accelerates test execution by leveraging vertical specific automation / point solutions
Test reporting	• Go and No-Go decision based on business criticality of the unresolved defects

Let us now consider certain situations to understand better how domain/vertical knowledge can help QA organizations ensure 100% defect free deployment and adherence to stringent schedules.

Situation 1 – OSI Compliance Testing for US Banks

OSI was introduced by US regulatory bodies to increase the option symbol length, from the earlier 5 characters to 21 characters, to manage the expected growth in options volumes, as well as to enable introduction of new products. This was a federal mandate that impacted all US Banks and Financial institutions, and the compliance date was common, thus making it non-negotiable. The Option Clearing Corporation (OCC) also published various guidelines which required the banks' and financial institutes' IT teams to have a thorough understanding of Option industry functions. Given the high stakes involved due to this mandate, extensive end-to-end regression testing for all combination of options and order types was necessary. Further, in a majority of the financial institutions and banks, changes impacted many applications in IT systems.

In such a scenario, what would the leadership of financial institutes really want from the IT organization? The answer is pretty simple. They would want the IT team to deliver specialists who understood the OCC guidelines and the impact these would have on the IT infrastructure and on the overall business itself. Domain

iseb Accredited Training Organisation

1. akkreditiertes Unternehmen im deutschsprachigen Raum

ISEB Intermediate (deutsch)

Der ISEB Intermediate Kurs ist das Bindeglied zwischen dem ISTQB Certified Tester Foundation Level und dem Advanced Level. Er erweitert die Inhalte des Foundation Levels, ohne dass man sich bereits für eine Spezialisierung - Test Management, technisches Testen oder funktionales Testen - entscheiden muss. In drei Tagen werden Reviews, risikobasiertes Testen, Test Management und Testanalyse vertieft; zahlreiche Übungsbeispiele erlauben die direkte Anwendung des Gelernten.

Eine einstündige Prüfung mit ca. 25 szenario-basierten Fragen schließt den Kurs ab. Das „**ISEB Intermediate Certificate in Software Testing**“ erhält man ab 60% korrekter Antworten.

Voraussetzungen

Für die Zulassung zur Prüfung zum “Intermediate Certificate in Software Testing“ muss der Teilnehmer die Prüfung zum Certified Tester Foundation Level (ISEB/ISTQB) bestanden haben UND entweder mindestens 18 Monate Erfahrung im Bereich Software Testing ODER den akkreditierten Trainingskurs “ISEB Intermediate” abgeschlossen haben - vorzugsweise alle drei Anforderungen.

Termine

08.08.11–10.08.11	Berlin
21.11.11–23.11.11	Mödling/Österreich

€1600,00

plus Prüfungsgebühr €200 zzgl. MwSt.

http://training.diazhilterscheid.com

Díaz Hilterscheid

knowledge would not only help these specialists implement static testing, which helps in early validation, but also allow them to work as catalysts to identify the impacted systems in the client's IT and business landscape. This would help expedite the process of testing and deployment. Further, with proper domain enablement, the QA organization would be able to use ready-made options trading test case repositories and automated test libraries that could accelerate the building of the much needed end-to-end test scenarios for the proposed OSI changes. These steps would help QA teams ensure 100% coverage and faster implementation of the proposed OSI changes without losing much sleep over it.

Thus we can see how the knowledge of a particular domain, in this case OCC guidelines, would have made the process of compliance for banks and financial institutes a lot easier and smoother. More importantly, it would be a lot more hassle-free. The same applies to other industries, too.

Situation 2 - 5010 Compliance validation for large US based insurance companies

The Health Insurance Portability and Accountability Act (HIPAA) requires the adoption of X12 version 5010 (migrating from 4010/4010A standards), which is to be used by covered entities (health plans, health care clearinghouses, and certain health care providers) when they conduct certain health care administrative transactions electronically, such as claims, remittance, eligibility, claims status requests and responses. A 5010 upgrade has 968 unique changes, as compared to 4010, across all nine X12 transactions supported by clients, and this demands that testers be well-versed with all impacted areas. Further, like every other regulatory compliance, the 5010 compliance mandate comes with very strict deadlines forcing insurance companies to shorten implementation timelines significantly.

In such a scenario, to address the need for effective and accelerated testing, the insurance company needs to have -

1. A Testing solution or accelerator that can help test the data conversion from 4010 format to 5010 format. The conversion can be a very data intensive process and its testing may require that appropriate business rules and configuration be made available in advance to ensure testing proceeds with minimal customization.

2. Ready-made test repositories containing 5010 files across all 9X12 transactions. This will provide a good starting point for the creation of test scenarios and ensure optimal coverage.

3. An effective training programme to share domain specific knowledge and availability of a pool of experts to address sudden ramp-up requests by business teams looking to accelerate business compliance

So, from the above examples it is evident that an understanding of domains certainly helps QA organizations to respond to business requests more effectively and efficiently. Having said that, does understanding of business/domains help only in compliance situations, or is the impact felt in the day-to-day functioning of QA organizations too? Let's look at another situation which will help us answer this question.

Situation 3 - E-Commerce Testing Solution for retailers

In the last decade or so, the landscape of software applications within organizations has undergone a fundamental change. In fact, the retail industry has witnessed these changes more than most other industries. The democratization of software applications has relegated the single, monolithic, restricted-user applications of yesteryears to history. Today, the IT landscape in the retail industry is filled with software applications that are dynamic in nature, supporting thousands, sometimes millions, of users at any given point in time. Most importantly, IT applications in the retail industry support revenue-generating activities and are thus crucial to the company's revenues.

All these changes have made the process of testing applications a lot more complicated. Let's take a simple example of an e-commerce program aimed at integrating multiple e-commerce websites of the retailer onto a single platform. In such a case, the system faces various challenges, as it is required to:

- support end users to complete business transactions faster irrespective of the user's physical location or the mode of accessing the e-com portal.
- provide the user with a secure shopping environment
- support the internationalization requirement
- support a large diversified, distributed and differently enabled user base

Reusable pre-defined test scenarios modeled around standard e-com business processes can help ensure faster time-to-market, which is a key business imperative for any retailer. Hence it is very important for QA teams to be knowledgeable about typical security, load, usability and accessibility test scenarios which need to be tested during the implementation of e-commerce portals. Security testing especially becomes very important, even more so than the additional features an e-com portal may provide, as online shoppers are concerned about security of their personal information. Previous knowledge of typical security vulnerabilities in similar applications goes a long way in creating specific test scenarios that can help de-risk new implementations. Further, domain experts play a critical role in developing ready-made point solutions / scripts which help accelerate the test execution, thereby reducing overall implementation time and effort.

So, while it may not be too hard to appreciate the value of vertical/domain focus in QA, the challenge lies in achieving it. The simplest way to achieve this could be training all team members on the domain, but is that enough?

The key in making the vertical focus sustainable and effective is to identify ways of integrating it in the day-to-day activities of the QA team. Reusable assets comprising of test scenarios covering commonly implemented business processes for each business vertical, pre-built automated scripts for industry specific ERP packages, and test cases built from a repository of commonly encountered issues (based on end-user feedback) are good examples of how a vertical focus can be built into QA practices, thus making them both efficient and effective.

Conclusion

Verticalization of QA is already helping bridge the gap between the business users and the IT teams thus helping increase the relevance of the work product that is delivered to the end user. The vertical specialization is creating a platform for innovation that will shape the way QA practices will evolve in times to come. This focus provides a unique solution to address the triple constraint of cost, quality and time that the QA teams have always been challenged with. The QA organizations should rapidly embrace this change and focus on creating a competency development process to ensure that their teams stay abreast of the latest trends and business requirements, in order to continue delivering on the promise of certifying applications that are ready to deploy and meet the end user's requirements.

> biography

Shishank Gupta
is a Delivery Manger and leads the Retail division within Infosys' Independent and Validation Practice. He can be reached at shishankg@infosys.com.

Rajneesh Malviya
is a Senior Delivery Manager and leads the Banking and Financial division within Infosys' Independent and Validation Practice. He can be reached at rkmalviya@infosys.com.

Experiences using TMMi® as a Process Reference Model for Test Assessments

by Matthias Rasking & Simon Lamers

An overview of the recently completed TMMi levels and experiences using an assessment framework for test assessments.

Efficient and effective processes are a cornerstone for each testing organization, yet few seem to have transparent view of the maturity and efficiency across projects and applications. Moreover, key questions such as "Where do I start improving?", "How can I establish the changes so that they are part of our culture?" and "How do I know that the improvements actually have an impact on my bottom line?" need to be answered before starting with a process improvement journey, otherwise experience has shown that a lot of paper is being produced without real change. In order to deliver real impact through process improvement activities, it is important to understand the structure of the underlying process framework and how to use this within each specific situation. Many useful, but proprietary (or linked to individual persons) process improvement models exist for testing processes. These, however, fail to address the need for an industry-wide and open standard that is continuously maintained and improved.

The development of TMMi

The Testing Maturity Model Integration (TMMi) serves the purpose of defining testing processes in a transparent, open and standardized way. Originally developed as TMM by the Illinois Institute of Technology, the TMMi Foundation, a not-for-profit organization based in Ireland finished this work in December 2010 with completing the top maturity levels 4 "Measured" and 5 "Optimized". Since everyone is welcome to join the TMMi Foundation as an individual member (or a corporate sponsor), the completion of the final maturity levels also was a collaborative effort by a team consisting of members from Accenture, AppLabs, Cognizant, Experimentus, the Hong Kong Polytechnic University, Improve QS, SysQA und Wipro. The ongoing development led to an internationally accepted standard for a Test Process Maturity Assessment Model, including an assessor's scheme and qualification requirements.

For more details on levels 1 to 3 and basic information on TMMi, we recommend two articles from a previous Testing Experience issue (03/2008): "Test Process Improvement using TMMi" (by Erik van Veenendaal, Rob Hendriks, Jurian van de Laar and Bart Bouwers) and "The Test Maturity Model Integrated (TMMi ®)" (by Brian Wells).

Overview of maturity levels 4 and 5

The recently published top maturity levels 4 and 5 again require the implementation of any previous maturity level. Therefore all test processes have to be standardized organization wide and a technical, managerial, and staffing infrastructure capable of thorough testing and providing support for test process improvement has to be put in place as defined for maturity level 3, "Managed". In order for the measurements to allow meaningful analysis of processes, the processes in question need to have been established across the organization in a consistent manner as defined by maturity level 3. While it is possible to measure individual instances of processes, the value of management by measurement really comes to life when being able to compare these measurements across parts of the organization. Having your organization and projects running on this level, the testing processes now can advance to TMMi levels 4 "Measured" and 5 "Optimized".

Maturity Level 4 "Measured"

At level 4, quality and process efficiency are being statistically measured throughout the whole development lifecycle and the test process is being managed using this information. The process area "Test measurement" focuses on how to identify, collect, analyze and apply measurement activities for test processes on an organizational level, but also on how projects can make use of the collected data (e.g., for objective planning and estimation). This is an evolution of the level 2 process area "Test Monitoring and Control", where monitoring progress and quality against the plan has been defined in that it includes practices on data collection and analysis in addition to a holistic view of test measurements. Also covered by this area are test performance indicators specified in process area "Test Policy and Strategy" on level 2 and the generic practice "Collect Improvement information" on level 3. On the other hand level 4 process area "Product Quality Evaluation" considers the measurement of the product under test on a project level. It describes how to identify product quality needs, how to define the quantitative product quality goals and how to measure them throughout the software development lifecycle. It builds on level 2 process areas "Test Design and Execution" and "Test Monitoring and Control" as well as "Non-functional Testing" from level 3. The third process area "Advanced Reviews" supports this by measuring (product) quality as early as possible in the

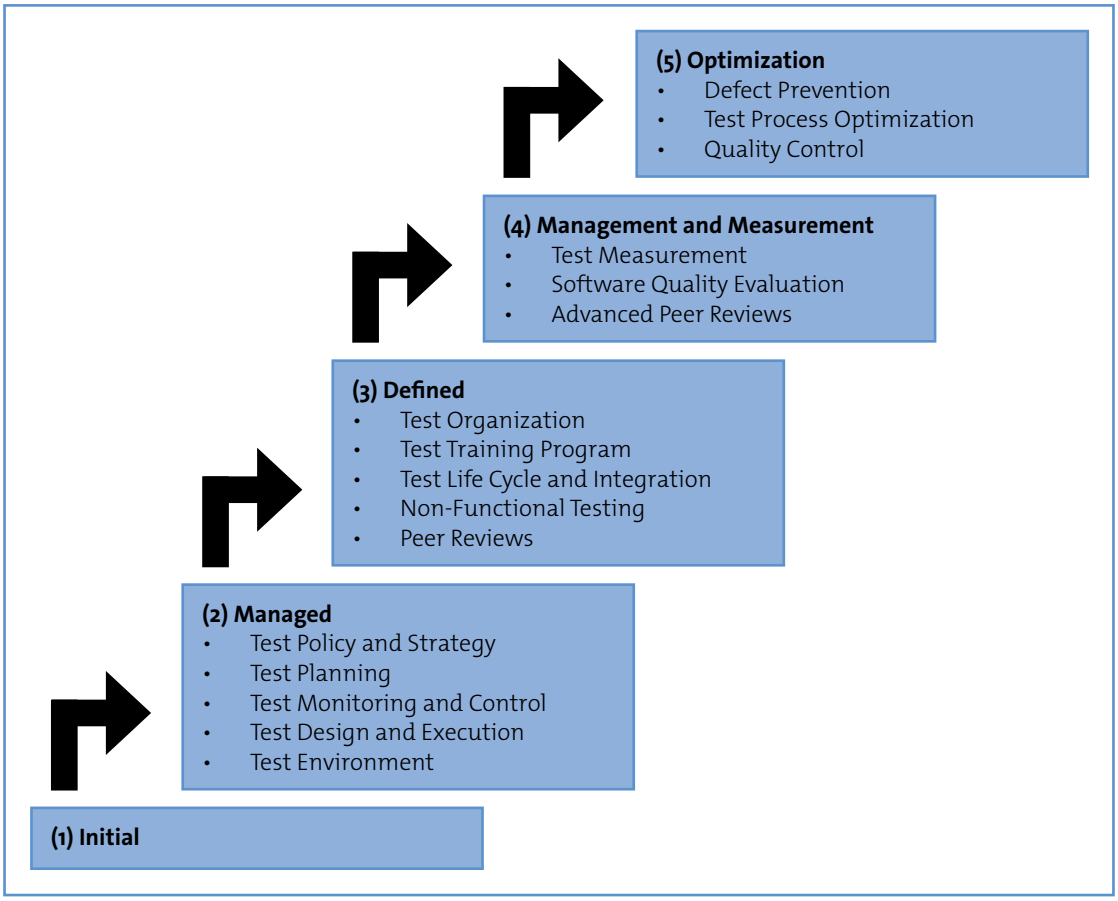


Figure 1: Overview of TMMi Maturity Levels

lifecycle. While "Peer Reviews" (introduced on level 3) are mainly performed to find defects, the review practices are now being enhanced to include practices like sampling, applying exit criteria, and prescribing rules.

Maturity Level 5 "Optimized"

After having achieved all improvement goals at levels 1 through 4, testing processes are completely defined and measurable. Based on the measured information (as specified by level 4 measurement practices) an organization is now capable of continually improving its processes according to the level 5 process area "Test Process Optimization". Furthermore new testing technologies such as methods, techniques or tools are being integrated into the organization if appropriate. On the other hand analysis of data collected according to level 4 measurement practices can now show common causes of defects allowing the organization to define actions to prevent similar defects from occurring in the future as defined in the new process area "Defect Prevention". Having implemented level 5 process area "Quality Control" and the required and related areas "Product Quality Evaluation" and "Test measurement" from level 4, the effectiveness of a test process even becomes predictable because its defined measures are statistically controlled. Process outcomes other than expected are variations due to human error, influences outside of the process or other hardly predictable events and can be used to identify further improvement opportunities in existing processes.

Process Improvement with TMMi

After having introduced the two new maturity levels, the question now is: How can an organization get there? First of all the initial state of the testing processes has to be determined. The TMMi reference model can be used both to define the initial state and the target level of testing processes. The transparent structure, which is aligned with CMMI and ISO/IEC 15504, allows IT Leads and Test Managers to define the current state of process maturity within their area of responsibility by conducting an As-Is-Analysis of their current processes against the specific goals defined in TMMi. A way to facilitate this assessment is Accenture's patent-pending Test Assessment Framework, which can serve as a questionnaire to elicit the individual practices and consolidate them across units.

Stakeholder perception: "We have a Test Policy in place. I approved it a couple of years ago."

Process reality: "I have never read this policy, it's too high-level and my project has special circumstances so that I cannot apply the policy."

Stakeholder perception: "We achieve 90% test coverage each release."
Process reality: "We execute 90% of defined test cases – but we have no idea whether the 100% of test cases cover 100% of the requirements."

Stakeholder perception: "All defects are being managed centrally in a tool."

Process reality: "The documentation of defects varies greatly depending on the project, a lot of clarification mails are necessary, and we cannot use the defect data to move towards defect prevention."

An assessment's potential pitfalls

As evidenced above, an assessment is therefore a question of perception. The statements above show the subjective current state of testing processes as perceived by various stakeholders compared to the results of a TMMi-based appraisal carried out with

the same stakeholders and some of their team members. Obviously the stakeholders believed to be operating at a much higher level of proficiency already, but how can this happen? Surely everybody should understand process maturity and be realistic enough regarding their own process performance? Let's take a look at some common pitfalls for wrong perceptions and failed expectations towards the outcome of a test process maturity assessment:

1. Understanding of the process itself:

We see this over and over again – what "test design" means to one person can be a completely different deliverable or process to someone else. While TMMi provides a foundation for these process areas, it depends heavily on the knowledge and experience of the person being asked about process performance to adequately assign a maturity level. Remember that the TMMi appraisal will validate whether each goal of the process area has been met, whereas the stakeholder might simply think about how "Test Design" is being performed currently without taking all the detailed goals and practices into account. This is very similar to estimating work plans and project efforts – unless you very clearly define the scope in terms of deliverables and expected process steps, the estimate will always depend on the viewpoint of the estimator.

2. Filtered information on process performance:

Depending on the level of stakeholders, it could potentially be a problem that the information they are getting on process performance is being filtered by their direct reports. They might get isolated issue reports and "lessons learned" accounts on processes not working, but due to problem #3 and the inclination of their direct reports to report everything as green (or maximum yellow/orange, but never red), they might not have the full picture and are therefore more confident.

3. Lack of verifiable, measureable process performance:

Measurable process performance already necessitates a test organization to be at maturity level 4 in TMMi since you first need reproducible processes across your organization to effectively measure the performance across the organization. Therefore many organizations simply do not have the means to objectively evaluate process performance and rather stick to individual opinions and process examples.

4. Insufficient appraisal information:

These "individual opinions" can also be a problem for the appraisal team itself. Depending on the experience of the appraisal team, the preparation done for the appraisal and the time available, the appraisal might be based on very selective evidence and therefore not show the whole picture of process performance. Group interviews have proven to be a good way to get information, reducing the time necessary for everyone involved and fostering an atmosphere of open communication (make sure there are no direct reports in one group, the group should always consist of more or less the same hierarchy level). Get evidence before the interviews in order to be able to ask pointed questions, and keep the questions open-ended in order to not lead the interviewee down a path of answers.

These potential pitfalls illustrate the importance to support any process improvement initiatives with common and reference-able models, while using change management initiatives to make sure all stakeholders understand the assessment about to take place.

Based on the information gathered in the assessment interviews and through document (evidence) reviews, the specific practices, sub-practices and examples contained in TMMi can then be used to elaborate on obvious improvement potential and provide the groundwork for the implementation of improvement recommendations. TMMi as an open standard allows all stakeholders (from responsible management to individual project members and finally the business and operations team impacted by poor quality or schedule performance) to analyze the impact of any improvement recommendations, prioritize them and adapt the change management journey to the applicable situation.

Which goal to set?

So, which maturity level will be the right one for an organization? There clearly can't be one simple answer to this question and even an assessment won't tell you. As mentioned above, what an assessment will show is where the organization has room for improvement, but the decision on what to improve has to me made by the relevant stakeholders within the organization. They should consider all factors which impact their everyday business: Do we only run a few small non-critical projects a year? Probably the goals at level 2 or 3 may be sufficient ("At least we know what testing is, how we want to test is well documented and we run these tests."). But when it comes to a broader project or application portfolio, at least the goals at level 3 should be considered. Especially for e.g. business critical applications, or if testing as a service is offered within the company or for external clients, the testing organization should definitely aim to satisfy the goals up to level 5. In other words: if you just want to establish a consistent and efficient way of testing in your organization, level 3 should usually be the first objective. Operating at this level may be ok for the time being and solve a lot of problems that may have occurred in the past. But staying at this level may mean stagnation because it is difficult to improve effectively and efficiently without the measurement data collected using practices at level 4 and using this data for continuous improvement as defined in practices at level 5. Therefore even if achieving maturity level 3 is the goal, the test organization should take a look at the goals and practices at levels 4 and 5 for additional guidance and objectives.

Planning a Process Improvement Project

Process improvement projects based on these improvement recommendations should not be taken lightly, but set up and planned as a separate project. This project should consist of several phases or work packages so that the implementation of each recommendation can be measured and done relatively independently of ongoing work. Remember that each improvement recommendation is (productive) criticism of the current status quo. Therefore each characterization of current behavior should be done on the basis of processes, not people in order to achieve broad buy-in to the recommended changes. Using the assessment results as an indicator of current behavior helps with defining gaps against the TMMi goals as well as the goals (vision, mission) for the company itself. If no such test policy and mission statement of testing exists, this should be the first action item for the process improvement team. Existing gaps and strengths can be identified and utilized to define concrete objectives, e.g. quality improvement, cost reduction, employee and/or customer satisfaction, and reduction of time-to-market. The specific practices within TMMi can then be put into individual work packages

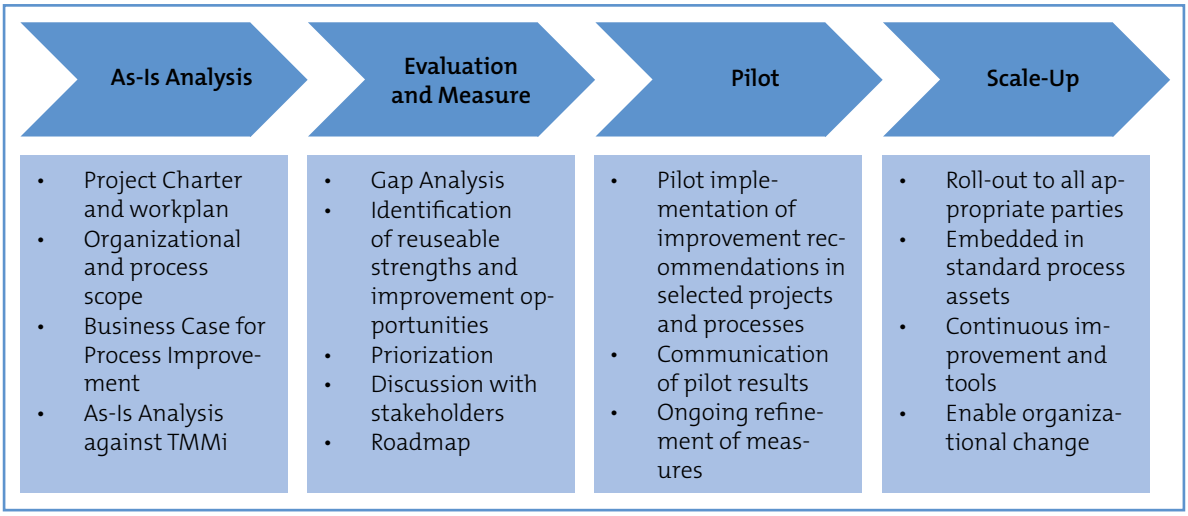


Figure 2 – Process Improvement Project phases

for the process improvement team.

After this thorough analysis of the work packages they can be verified in pilot projects and refined based on the actual results to roll out their implementation to other parts of the test organization. Supporting training courses, change management measures and communication are very important in this step in order to gain acceptance for these changes and make them understandable to a broader audience.

Conclusion

During the last years and not only with completion of the final maturity levels, TMMi became the open standard for process improvement the Quality and Testing industry needs. It solves the dependency problem of other models by using an open and collaborative method to include various viewpoints and can therefore help test organizations with improving the effectiveness and efficiency of their testing processes.

> biography

Matthias Rasking
leads Accenture Test Services in the German-speaking markets as well as Accenture's global Testing Community of Practice of over 13,000 testing practitioners. With more than 10 years of experience in the Testing and Quality Assurance space, Mr. Rasking holds or has held certifications in software engineering practices such as CMMI, ITIL and as an IEEE Certified Software Development Professional. Mr. Rasking is the Working Group Manager for Model Development and Maintenance at the TMMi foundation. He has supported many multi-national clients in various industries in becoming high-performing businesses by establishing a structured and strategic approach to quality assurance.

Simon Lamers
works as a consultant for Accenture Test Services in Germany and focuses on test management and requirements engineering in complex IT projects. With more than 7 years of experience in the Testing and Quality Assurance space, Mr. Lamers holds certifications in software engineering practices, e.g. from the ISTQB and ITIL. He is also a member of the TMMi working group for Model Development and Maintenance.

A DARN good process to evaluate and purchase test management systems

by Harish Narayan & Igor Ushakov

Do you have questions on how to evaluate and recommend the right test management tool suite for your organization? Are you looking for specific steps and techniques to use to guide a well-informed decision? If so, this article will be of value to you. We will share a DARN good (best-in-class) process for how to accomplish this, one that should include buy-in from all test and management stakeholders. What is **DARN**? It stands for **Define, Assess, Re-examine and Negotiate**. It is a process that can be used to guide any technical evaluation and purchase, especially one that has broad organizational impact over many years. In this article, we will detail this process along with a real-world illustration of how we used it to effectively recommend and purchase a test management system for our organization.

We will start with a quick overview of the four (4) phases in the process. They are as follows:

- **Define**: This phase ensures that all stakeholders are heard and is a critical step to ensure that the evaluation is objective and any subsequent adoption has organizational buy-in. It is imperative to first gather requirements from all stakeholders, perform a needs analysis (must have, desirable, nice to have, out of scope), validate this and define the use case scenarios for the evaluation. The phase ends with the selection of a list of tools or systems to be evaluated, and provides an overview of the scope of evaluation to management.
- **Assess**: Integral to this phase are three (3) areas of analysis – quality assessment, cost analysis and organizational considerations. At Vistaprint, our quality assessment process is called QODA (Quality-Oriented Decision Assessment). It is based on the Lightweight Architectural Alternative Assessment Method. It is a best practice used to analytically categorize, rate and rank various tools or systems under evaluation based on use case scenarios. It answers the questions, "*what characteristics would make a tool or solution a good choice?*" and "*how important is each of those qualities, in relation to each other?*". This objective evaluation coupled with cost analysis and management verbal commitment is used to narrow down the choice to two tools or systems.
- **Re-examine**: This phase of the evaluation involves a 'bake-off' of the two tools or systems, along three (3) key dimensions – user evaluation against requirements, integration evaluation with automation, and integration evaluation with other tools used in the organization, like a defect management system. The findings from this evaluation and a definitive cost analysis are used to recommend a tool suite for purchase.
- **Negotiate**: Once final management approval has been obtained, the key to finally 'closing' the deal is to negotiate various aspects needed to gain adoption and protect the investment. These include not only licenses, support and maintenance, but also professional services and training.

First, some background information for context. We have an internally developed Test Case Repository (TCR) that was used by only one part of the QA organization. There was cultural resistance to adoption in other parts. Secondly, the tool, though well designed, was more aligned to work with the automation processes, not all of the broad QA processes. So, there was a clear need to either extend the existing in-house tool or buy and deploy a well-known third-party tool.

Before moving into the define phase, we ensured two things were in place. One, the evaluation and subsequent extension or deployment of such a system was defined as an initiative and published as part of the organizational 2-year roadmap. This was then broadly communicated across the technology organization. This usually ensures gaining mind share across various teams in the organization and visibly shows management sponsorship. Next, we established a clear timeline for the initiative and defined milestones after each phase, to communicate progress and get periodic feedback to refine any steps in the evaluation. It is critical to run such an initiative as a project, with a defined plan that is updated frequently and has enough detail, so that things do not slip through the cracks.

Define – Clarify objectives and finalize requirements for assessment

A test management system offers greater visibility and transparency into test assets, coverage, and results, which leads to better alignment and informed decision-making on projects and releases. Our objectives for the test management system were as follows:

- Support test planning, execution, reporting and manage-

Description	Initiative	Priority	Stakeholders
Ability to group test cases by user defined attributes (multiple taxonomies, webpage flow)	TMS	Must Have	GD QA
Ability to store test cases of all types (functional, load/performance, use cases)	TMS	Must Have	All
Ability to combine test cases to form sequential scenarios	TMS	Must Have	GD QA
Ability to move test cases across taxonomies, teams, etc.	TMS	Must Have	All
Tool should support user authorization (user/admin roles)	TMS	Nice to Have	GD QA
Ability to easily update test cases (group edit)	TMS	Nice to Have	All
Ability to attach files to test cases	TMS	Must Have	FD QA
Ability to use rich text editor for test case editing	TMS	Desirable	All
Ability to search / sort / filter test cases by multiple attributes	TMS	Must Have	GD QA
Ability to export test cases	TMS	Desirable	GD QA
Ability to support multiple users working at the same time	TMS	Out of Scope	FD QA
Ability to import test cases into the tool	TMS	Desirable	FD QA
Ability to store test cases types that are a hybrid of automation and manual	TMS	Must Have	FD QA
Ability to link from wiki to tool at test case level	TMS	Desirable	All

Figure 1: Needs analysis

ment on projects and releases

- Be able to manage test basis for both manual and automated testing
- It should scale and effectively support growth, of both test cases and the business.

Keeping the high-level objectives simple helps further mind share in the organization. It becomes an 'elevator speech' and a value statement.

Every QA manager identified participants for requirements gathering. Requirements or needs were gathered through group discussions and captured first. We then categorized these requirements into the following:

- Must have – should be available in the first version or initial deployment
- Desirable – should be in the roadmap for the second version or extension
- Nice to have – would add to stakeholder delight but is not required
- Out of scope – not required or will not be in the deployed version

Figure 1 shows an illustrative capture of the needs analysis. TMS stands for Test Management System. The stakeholders' column was used to capture which QA team initially stated the need or requirement. This is useful later in any evaluation to be able to demonstrate '*which tool fits best?*' and to answer the question, '*what's in it for me?*'.

We refined the requirements into use case scenarios. A use case format is helpful in adding context to requirements and to better structure technical evaluations. Given that there are so many test management systems in the market, we short listed a few for the next phase based on a few key considerations. Since one of the objectives during assessment was to decide whether to buy or extend our in-house system, the in-house tool, TCR was included. We chose four other third-party tools based on feature set, market share, adoption rate in similar industries, deployment options, ability to integrate with other tools, their ability to be deployed on our technology infrastructure and cost of ownership. We reiterated the scope of the evaluation with management and moved on to the next phase.

Assess – Evaluate and decide on the tools to re-examine further

This phase includes both a quantitative and qualitative assessment, and a preliminary cost analysis. QODA (Quality-Oriented Decision Assessment) was used for the objective quantitative evaluation of the various tool alternatives. When a decision needs to be made with multiple alternatives, the optimal solution is usually not immediately evident. When these situations surface, a common approach is to list the pros and cons of each alternative. The belief, given such a list, is that one will be able to compare and contrast the qualities of each choice in order to make a good value judgment. However, there are multiple reasons why creating a simple list of pros and cons is not optimal. Comparing the pros and cons of each choice in aggregate blurs the importance of each individual feature or use case. Cons of one choice are often listed as the lack of pros from another choice, resulting in a double-counting effect. Rather than comparing the available choices to each other, it is better to determine the qualities that would make up an optimal choice. The available choices can then be scored independently to determine the fit. The QODA process we used to evaluate the test management tools has six (6) steps:

1. **Construct the Quality Tree:** A „Quality Tree" is the organizing mechanism for any QODA. The tree has 3 nodes – quality groups or level 1 qualities, level 2 qualities and use case scenarios. Below the root of the tree, we capture the primary attributes that are relevant for an assessment. These level 1 qualities typically include the "ilities" that are familiar to us all, such as usability, operability, maintainability, etc. For our evaluation, we used operability, "integratability", configurability, usability, performance, maintainability, "deployability" and "securability". These level 1 qualities don't go far enough in terms of granularity. For instance, how do you determine if the system is maintainable or secure? Level 2 qualities are helpful in bridging the gap between our intuitive understanding and expanding to fully precise scenarios or use cases. The easiest way to construct the quality tree is to use requirements (use cases) gathered from stakeholders.

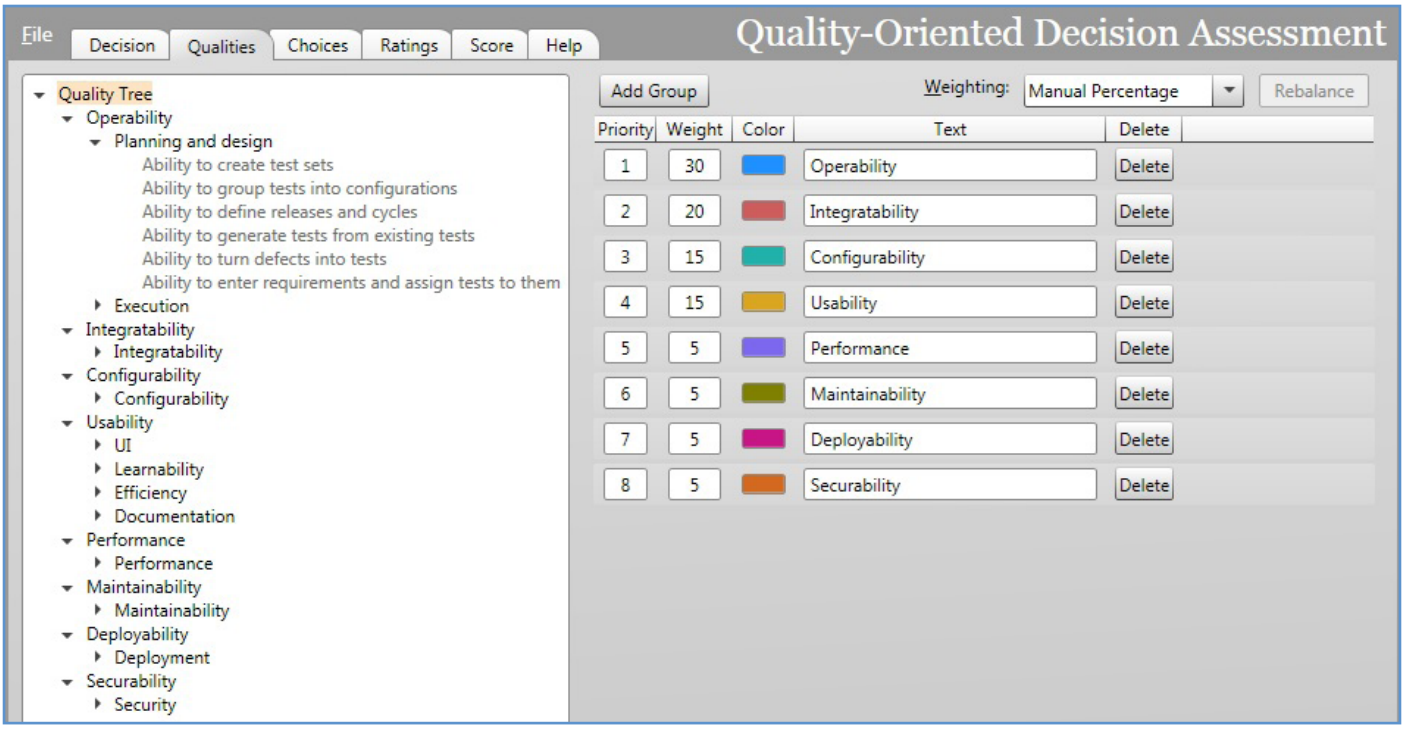


Figure 2: Example of a quality tree

They help generate a mix of scenarios and quality properties, which can be naturally organized into a hierarchical form. Figure 2 shows a portion of the quality tree used in the test management systems evaluation.

2. **Finalize Scenarios**: Use case scenarios are the most important part of QODA. They make qualities concrete and measurable, ensuring that we have something we can actually evaluate objectively. Answering questions about how well test management system alternatives achieve qualities, expressed as use case scenarios, is necessary and effective because scenarios provide a well-specified characterization of a behavior or property of a tool. Therefore, you have to ensure that all stakeholders sign-off on the scenarios before you proceed to the next step. Figure 2 shows an example of scenarios that we used under Operability – Planning and design.

3. **Rank Qualities and Scenarios**: The next step is to prioritize the members of the quality tree. There is a need to capture how important each quality attribute is to the stakeholders. Qualities are ranked hierarchically at each level of the tree. The level 1qualities are ranked first. Then the level 2 qualities are ranked within the context of each quality group. Subsequently, the use case scenarios are ranked within the context of each level 2 quality.

Ranking of qualities and scenarios is generally accomplished by actively collaborating with the stakeholders. Sometimes, however, it is not possible to achieve consensus and alternative approaches are required. We used one such approach in our evaluation. In this approach, we asked the representative from each stakeholder group to individually rank each quality and scenario. We used complex least squares computation to combine individual ranks and determine the final ranking.

In addition to using ranks, weights are also applied to each quality tree member. They are helpful in providing additional means of prioritizing the importance of each scenario.

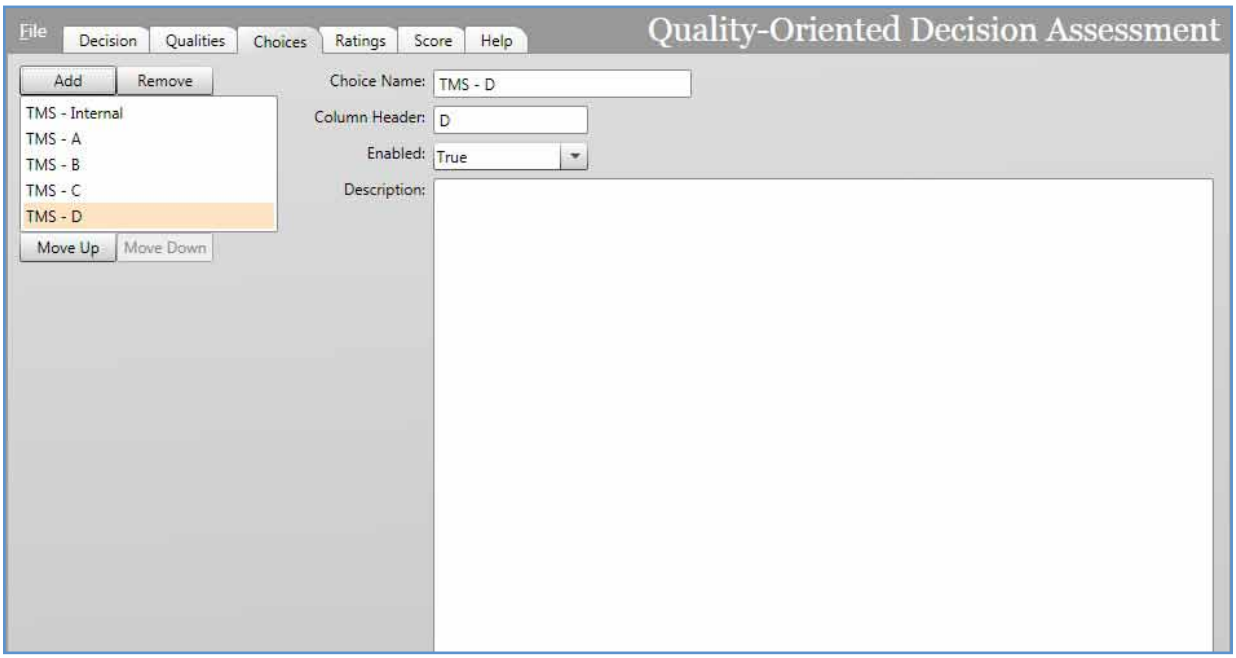


Figure 3: Test management system alternatives used in the QODA

Figure 2 shows an example of prioritized and weighted qualities. Operability, "integratability", configurability and usability made up 80% of our weighted assessment criteria.

4. **Define alternatives to be assessed**: The final inputs to QODA were the test management system alternatives to be assessed. We included our existing tool, TCR, as one of the alternatives because it was possible that none of the tools being considered offered significant benefit over the existing approach, and we also had to make a „build vs. buy" decision. Figure 3 shows the choices that we used in our assessment including our existing TMS (due to certain constraints, we have chosen not to show the actual names of the third-party tools we considered).
5. **Rate alternatives**: The next step is to assess each alternative against each use case scenario. Figure 4 shows a ratings snapshot of our assessment of test management systems. When assessing against a scenario, each alternative gets a rating on a five point scale. They are the following:

- None - The tool fails to meet the quality requirements of the scenario
- Poor - The tool does support the quality, but it is less than acceptable
- Adequate - The tool meets the minimum need, and is considered acceptable
- Good - The tool more than adequately meets the requirements of the scenario
- Excellent - The tool is outstanding for this scenario.

It is important to rate the choices uniformly. That is, if one choice is rated for a particular scenario, then the remaining choices must also be rated. Not all scenarios need to be rated, but it is a good idea to work top-down, and rating more scenarios produces a more comprehensive assessment score.

File | Decision | Qualities | Choices | Ratings | Score | Help

Quality-Oriented Decision Assessment

Internal	A	B	C	D	Text
Poor	Good	Good	Good	Good	Ability to group test by user defined attributes
Poor	Good	Poor	None	Good	Ability to define test parameters
Good	Adequate	Adequate	Poor	Adequate	Ability to integrate with automation
Poor	Good	Good	Good	Good	Ability to create test sets
Poor	Good	Poor	Good	Good	Ability to define releases and cycles
Good	Excellent	Poor	None	Excellent	Ability to group tests into configurations
None	Good	None	None	None	Ability to support different test types / templates
Good	Adequate	Adequate	Adequate	Adequate	Ability to import existing tests
Good	Good	Good	Good	Good	Access to data
Good	Poor	None	Good	Adequate	Ability to integrate with defect management system (Jira)
Good	Good	None	None	Good	SQL Server support
Adequate	Excellent	Good	Adequate	Adequate	Ability to execute test sets (test runs)
Poor	Adequate	Adequate	Poor	Adequate	Ability to support data driven test
Good	Adequate	Poor	Good	Good	Ability to quickly respond to user input
Poor	Poor	Good	Good	Adequate	Authorization (Role Based Access Control)
None	Adequate	None	Poor	Adequate	Ability to display expected to actual test results
Good	Good	Good	Good	Good	Ability to install as a web package
Good	Good	Good	Good	Good	Authentication
Good	Good	Good	Good	Good	Ability to report test progress metrics
Good	Good	Good	Good	Good	Ability to report defect metrics
None	Adequate	Adequate	Adequate	Adequate	Ability to integrate with requirements management system
					The performance shall scale

Figure 5: QODA score

6. **Interpret results and determine next steps**: The QODA tool provides a final score based on the weighted priorities and values from the ratings matrix. This numeric score represents the relative quantitative assessment of each system. Figure 5 displays the QODA output. It is easy to see that 2 choices (TMS – A and TMS – D) clearly outperformed the rest of the tools.

Based on the QODA score, we performed a cost analysis on TMS – A, TMS – D and TCR. The total cost of ownership (TCO) analysis included software, maintenance, hardware and labor support costs for five (5) years. For the third-party tools, this analysis also included an estimate of incremental licenses needed in years 2 through 5. TCO for TMS - A was around $800K, of which labor cost was around $200K. The cost to extend and maintain TCR was mainly labor and was around $800K . TCO for TMS – D was around $500K. It is important to be as definitive as possible with this estimate. You should work with the vendors and internal teams, IT and finance, to come up with a comprehensive analysis. An illustration of one of the cost analyses is shown in Figure 6.

	2011	2012	2013	2014	2015	2016	2017
S/W License	150,000	28,000	32,000	40,000	44,000		
Support	35,000	42,000	50,000	60,000	71,000		
H/W	45,000	0	0	0	0		
Professional Services	20,000						
Cash Out	250,000	70,000	82,000	100,000	115,000		
P&L Impact	**Yr 1**	**Yr 2**	**Yr 3**	**Yr 4**	**Yr 5**	**Yr 6**	**Yr 7**
S/W	50,000	59,333	70,000	33,333	38,667	28,000	14,667
Support	35,000	42,000	50,000	60,000	71,000	0	0
H/W	15,000	15,000	15,000	0	0	0	0
Professional Services	20,000						
Total Impact	120,000	116,333	135,000	93,333	109,667	28,000	14,667

Figure 6 – Cost analysis including P & L impact

Organizational considerations made up the qualitative part of the assessment. Factors that favored updating our existing tool, TCR, included the following – our culture that favored building tools versus buying, the flexibility it would give us to customize, the agility with which we could extend and support the tool, and potential challenges third-party tools may have in integrating with our automation and defect management system. Factors that favored buying and deploying a tool included the following – these companies had years of product experience in test management and other software development tools, they provided an already rich set of features, they could be extended to support other aspects of the software development lifecycle, they provided support globally and had broader mind share buy-in across our QA organization.

After discussion with stakeholders, we recommended buying and deploying a test management tool based on the QODA score and qualitative analysis, which better aligned with the objectives we had defined initially. We chose TMS – A and TMS – D for further hands-on evaluation. We also obtained budget approval from management based on the cost analysis. Obtaining this approval is critical, since many evaluations never culminate in a deployment due to lack of financial commitment. In addition, without such commitment, tools are deployed less than optimally, are never adopted broadly and do not realize their full value.

Re-examine: Evaluate tools in detail and recommend one for purchase

In this phase, we evaluated the two third-party tools across three (3) dimensions. They were the following:

- **User evaluation against requirements**: A 6-week assessment by QA engineers and leads in the organization representing the interests of all the QA teams. This evaluation focused on using the tools and assessing whether the tools were learnable, flexible and easy to use to write tests, design, create and execute test suites.
- **Integration evaluation with automation**: A technical hands-on assessment to validate that each of these test management tools could integrate seamlessly with our automation frameworks, to help drive and then collect status of automation runs.
- **Integration evaluation with our defect management system**: A technical assessment to validate whether each of these tools could integrate seamlessly with our defect management system, so that defects entered in our system could be automatically synchronized into the test management tool. Both these integration evaluations were critical, as we intend to use the test management system to report not only on test case progress, but manual and automated test coverage, and quality (defect) metrics.

Before we started this phase, we took a few critical steps to ensure that not only the evaluation went well but that we potentially validated the needed system requirements to install and maintain these tools on-premise. We obtained 60-day evaluation licenses from the vendors with multiple user licenses instead of the usual 30 days, as our plan was not to just do a cursory evaluation but to have the users experience both the tools simultaneously while evaluating them against the defined scenarios. We defined a few core requirements for each vendor and had each vendor provide a 90-minute walkthrough demonstrating how their tool would handle the requirements. We also ensured that the vendors worked with any required partners for plug-ins or APIs needed to integrate their test management tool with our automation and defect management systems. Finally, we involved our IT operations and DBA teams to install the tools on-premise and validate that they would work on our technology stack. Documenting these steps in the project plan initially or as the evaluation progresses is crucial, as such steps, if not followed, may lead to delays and expectations not being met.

The user evaluation team prioritized and categorized the use cases, defined an evaluation scoring criteria and evaluated each tool. This team of seven recommended TMS – D over TMS – A by a 4:3 vote. Both the chosen test management systems had versatile APIs and we could validate that we could both drive our automation from these tools and reconcile results from our automation frameworks back into these tools. TMS - A provided a partner plug-in with which we could test bi-directional synchronization with our defect management system. TMS – D could only synchronize from their tool to our defect management system. It also worked only with a specific version of this defect system, which we did not plan to upgrade to anytime soon.

There was no clear winner between the two tools based on the evaluation across the three dimensions. This, in fact, reinforced our recommendation from the assessment phase to choose these two tools for the bake-off. Even though TCO was greater for TMS – A, we chose that tool for purchase and deployment as it better met the objectives laid out initially. The deciding factors were as follows:

- A rich feature set that would allow us to grow into the tool over time
- Its robustness as a tool and extensive global install base
- The rich API and ability to integrate with other systems
- Globally available 24 x 7 support.

Negotiate – Purchase the best possible options at optimum cost

We were able to work with the vendor to develop a quote that best suited our needs. In our case, an interesting option emerged out of the discussion. We also had other needs in our organization roadmap – one was to deploy a performance testing tool, and the other was to deploy a functional testing tool that supported a specific aspect of our testing. This test management system vendor also had industry leading tools in these areas. We negotiated a bundle that eventually gave us the following – more licenses for our test management system than originally planned; software, support and professional services to deploy a performance testing system; and software and support for a functional testing tool. The bundle will also yield us significant savings in support costs in years 2 to 5. In all, we accomplished all of this and were still very close to the budget approved by management.

We believe this phase is critical for a couple of key reasons. First, once management approval is obtained at a certain level, going way over or under budget will not be received favorably. It is why the cost analysis outlined as part of the assessment phase should be fairly comprehensive and definitive. Negotiating with the vendor knowing what you have internal approval for, makes the process go better for both sides. Second, deploying a new tool or system into the organization is as much an exercise in change management as it is a technical challenge. It should not be under estimated. Professional services to help with installation and customization will go a long way in accelerating adoption. Initial and ongoing training are also key drivers for not only adoption, but institutionalization for the long-term.

Summary

In this article, we shared an illustration of a DARN good process to evaluate and purchase test management systems. This process can be used as-is or modified to assess technical alternatives your organization may have to consider when making purchase decisions that could have long-term far-reaching impact. We would like to leave you with four (4) key takeaways, which are as follows:

- Scoping the work and involving relevant stakeholders from the beginning is important - to start off right, to guide decisions throughout the process, and to ensure higher probability of adoption post-purchase.
- Ensuring that the evaluation is objective, with clearly defined criteria and use cases, and both quantitative and qualitative, will significantly reduce the risk of making the wrong call.
- Continued management sponsorship throughout the process is critical for success. Clearly defined business value, supported by a thorough multi-year cost analysis and periodic communication will help in gaining and reinforcing management's confidence.
- Finally, building a good working relationship with the vendor of choice throughout the process and through the negotiation is critical. The vendor is a partner who is a key cog# in getting the system or tool deployed successfully.

> biography

Harish Narayan
is currently a director of technology at Vistaprint, focused on quality assurance and automation. He is a software engineering and information technology leader with over 15 years of experience. He has expertise in building and leading world-class quality assurance functions, and in championing enterprise-wide process and capability improvements. He is passionate about instilling a culture of quality in technology organizations and has been a proven business partner in diverse industries including telecommunications, financial services, e-commerce, enterprise software and consumer products. He also brings years of strategic planning, global operations, project management, performance management and team-building experience

Igor Ushakov
is currently a software engineer at Vistaprint and is leading the technical deployment of a new test management system within the organization. He is passionate about improving the quality of software whether he is implementing or automating it. Igor's interests include the development of automation, tools, metrics, and processes that can improve the overall quality of a product with effectiveness and efficiency. His expertise includes a master's degree in computer science which provides a solid basis for accomplishing his interests.

Communication up, barriers down – The key to success

by Juan Carlos Ocampo Calderón

The common belief is that having an against-all-odds process with a maturity level high enough to fulfill our expectations and needs, is what is needed to reach the success any organization needs. However, the key to this success does not rely on flow charts, or on a matrix of responsibilities, much less on the designers or the final approval they receive. The basis of any process is the people involved in it.

One of the main challenges companies face is breaking the barriers between the people and the process itself, and the most common problem is that your development and testing teams see themselves as completely opposite worlds instead of working together for a common goal.

Understanding the psychological origins

The typical system development life cycle takes us in a linear way from the software development process to its final stages, at which point we reach the testing process, and this is where two worlds come close.

On the one hand, we have the creative developers always ready to protect their creation from anyone trying to put it at risk, and on the other we have the tester who is in charge of validating that what the developers have produced fulfills the quality standards set by the user. More often than not, this task is seen by the developer as destructive, when in reality both need to understand that working together produces better results when delivering the final product to the user.

Improving the relationship

A crucial task for development leaders and testing teams is to pass a clear message and make sure each team member understands that their relationship should not be seen as a constant struggle, but as a collaborative effort for a common objective.

Every one from each team has knowledge and skills that are useful for certain tasks through the process. If the organization manages to combine these two working groups, the outcome will be better quality software in shorter periods of time and therefore lower costs.

Improving communication

Another important aspect to consider in this integration process is how situations and ideas are perceived due to the fact that the main problem often does not rely on what is said, but on how it is said. For example, if you find a flaw in one of the deliverables caused by a developer, the first thing the tester says is 'the deliverable is wrong' or 'the deliverable has a flaw', which surely will lead us to a negative scenario. We could change the result of this negative situation by portraying the defect or flaw in a positive way, for example, instead of saying that the deliverable has a flaw, it could be restated as a "chance of improvement or betterment".

The bottom of this lies in the fact that when people hear from another person that their work is wrong, they automatically take a defensive position, which could lead to conflict in the relationship. However, if we see the situation as a chance of improvement or betterment, it is more likely that the person will show openness to new ideas and further analyze the situation.

Effective Organization

A further point of improvement is to establish the strategic logistics for people management, specifically in the workplace. Three of the most popular models used by companies are listed below:

Shared workspace – It is considered that if development and testing teams share the same space, this can facilitate informal communication among the people, but it could mean trouble in monitoring and controlling defects or flaws and the corrections made on the objects tested.

Separate workspace - With this layout, teams are commonly found in separate offices. This situation helps to reduce substantially the informal communication between the teams, especially if, for example, development team and testing team are located on different floors of the building, or even in different buildings.

Monitoring and testing control are more efficient in this model, but the risk of special considerations between testers and developers, because of its good relationship, remains as high as in the previous model.

External testing team – Today, this is one of the most widely used models, due to the fact that it allows the separation of the testing and development teams by establishing total independence between the tester and the objects being tested.

It is considered that external testers are experts who have a high level of knowledge of the testing processes, and this ensures a suitable testing design, efficiency and the reduction of some costs.

Conclusion

All in all, we must consider the following aspects:

- The testing process is more efficient when there is good communication between teams.
- The documentation and description of the defects will determine the development of the artefacts during the testing process.
- The common goal should always be the main issue, which is delivering a high quality product to the final user and reducing time and costs.
- Externalizing testing in most cases increases the quality of the testing process.

> **biography**

Juan Carlos Ocampo Calderón *is a graduate in Computer Science and has been working as a tester in Quality Assurance for almost 3 years. He is part of the team which is responsible for the quality of the financial systems at the organization he works for in Mexico City.*
He is a Certified Tester of the ISTQB® Foundation Level and specializes in testing and quality improvement.

Shift Focus Towards Test Process Improvement For Business Transformation

by Kalyana Rao Konda

With the increasing need for efficient IT processes, growing customer expectations, and focus on high quality products, software testing has grown in prominence. In fact, software testing has become almost indispensible today as it helps in improving the overall quality of software or complex IT systems. With increased focus on cost and quality, most of the enterprises have been successful in establishing basic testing processes and a test team, but are often faced with several challenges. Though many enterprises invest heavily in testing, they often fail to reap benefits out of it. Most CIOs responsible for both development and testing often rue that despite making good investments in testing, end users often complain that there are too many defects in production, or they are concerned that testing did not raise the quality of their products.

Today, most enterprises have acknowledged that improving and establishing an efficient test process can solve many of these problems, but they often struggle to effectively define their current processes and have no idea on how to make the necessary improvements. Enterprises opt for test process consulting for various reasons, which are -

- Even after establishing a testing practice, enterprises fail to get enough test coverage or find defects too late in the process, ultimately increasing the costs to rectify them. Hence, enterprises want to check or benchmark their testing process with the industry standards.
- Enterprises focus on getting to the next level, but they don't know exactly how to go about it.
- Some of organizations wanted to just get a third party opinion on how well they are doing, and what they could do to improve. .
- Most of the stakeholders complain about the testing ineffectiveness
- Most of the customers complain about the defects in the end product
- To optimize the current testing process
- Current testing team is not able to deliver the expected ROI, etc.

The major reason for enterprises not reaping the benefits out of their testing practices is that most of them lack sufficient insight into the effectiveness and efficiency of their current testing process. This is clearly evident from the fact that, though enterprises adopt testing, they still incur high operating costs or may not be able to achieve the required quality levels or do not properly meet their business goals.

In general, some reasons why enterprises are faced with various testing challenges and fail to achieve a return on investment are,

- Testing is not properly aligned with the overall software development life cycle
- Testing is performed shortly before the production stage and often leads to rework and increased costs of fixing the defects late in the software development life cycle
- Lack of testing expertise leading to software going to production with lot of defects
- The testing process in place may be suitable for the current needs, but may not be suitable for the changing business requirements
- Their testing approach does not match the criticality of the system and is not properly aligned with development.

With the increasing competitive landscape and changing business requirements, testing and test process must be continuously monitored and improved accordingly in an organization. Test Process Improvement identifies the strong and weak points of the current situation in an organization and, based on the situation, suggests the necessary improvements. Basically, all test maturity models involve various activities such as test planning, test case design, execution, defect tracking etc. and help in assessing the software testing process.

What is Test Process Assessment?

Test Process Assessment is an approach or service that enables an organization to recognize and define current test processes, analyze those processes and identify areas for improvement. It involves the assessment of current testing processes and gives insights into whether they add any value to the project and whether they are being used or not. It helps organizations to gauge the maturity levels of the existing testing processes, provides clear insight into the current status of the testing maturity, and sug-

gests the necessary improvements required to improve the overall quality as well as contain costs.

Test Process Assessment also suggests the necessary steps required to implement the recommended improvements. It is done to potentially uncover what processes are working well and what are not going well according to the business needs. It differs based on the type of organization, such as large enterprises might be already adapting to well-recognized industry maturity models like TMMi or the TPI, while small enterprises may or may not have a proper testing practice in place. Often, there is a perception among small enterprises or with the decision makers of the small enterprises that process improvements are more apt to or mostly applicable for large organizations only. In reality, however, the test process improvements are equally applicable and important for all organizations, big or small, which are spending money on testing.

How to approach Test Process Improvement?

The implementation of any test process improvement takes place in a series of steps by taking into consideration a holistic view of the things in an organization,

- Speaking with sponsors and understanding business goals
- Assessing current test processes by performing reviews and conducting interviews
- Reporting assessment findings and establishing a roadmap for implementation
- Implementing recommended improvements
- Measuring post-implementation effectiveness

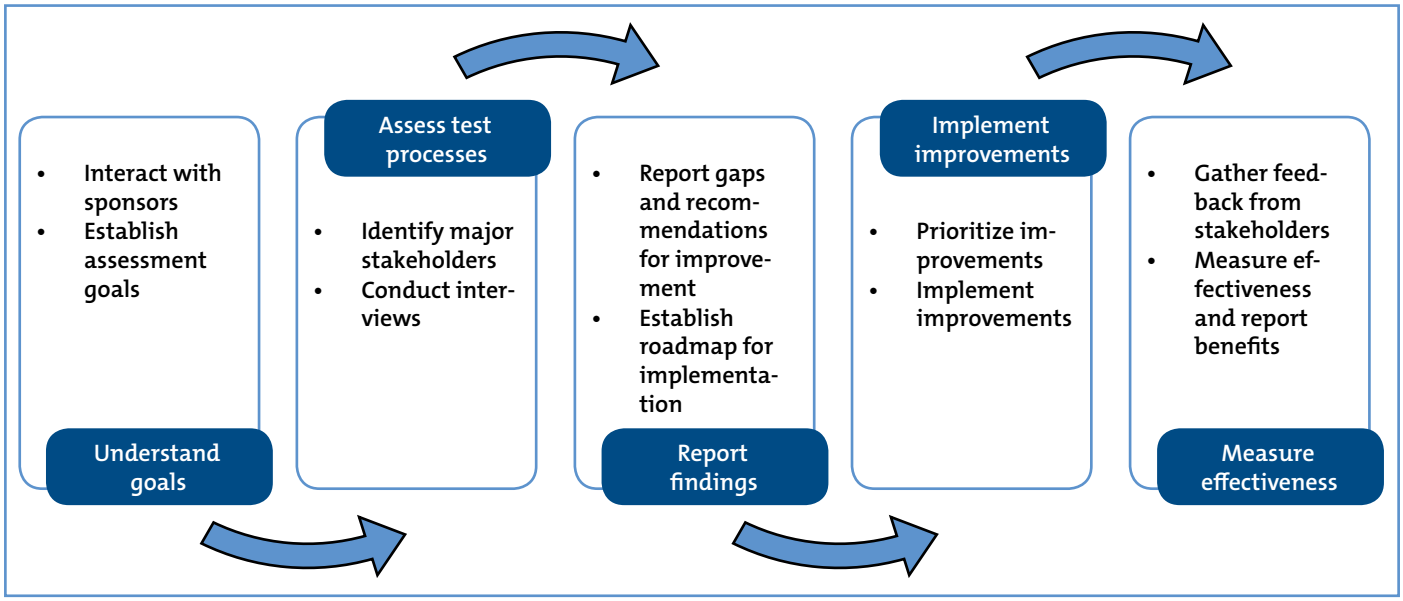


Figure 1: Five-step approach to Test Process Improvement

Step1: Speaking with sponsors and establishing assessment goals:

Sponsors are the people who assign the project, or take an initiative of assessing the test process, or sign the cheque for test process consulting. Hence, the first and foremost aspect of a TPC engagement is to talk to the sponsors, discuss with them in detail their organization's processes and the business goals. Many enterprises recognize that it is essential to improve their test process but don't have a clear idea or understanding of how to improve it or the necessary steps for improvements. After detailed discussion with sponsors, clearly defined goals for the test process engagement must be established. Irrespective of the method that is being implemented for test process assessment, the first and foremost consideration is to establish the goal of the assessment. Goals of the assessment can vary based on the type of the organization and its requirements such as,

- To minimize the cost of testing
- To improve the effectiveness of testing
- To improve the efficiency of testing
- To improve the overall testing maturity of the organization
- To raise the visibility of what testing contributes for the whole organization. etc.

Based on the requirements of an organization, clear goals must be established on what we are going to achieve as part of the test process assessment. Once the goals are established, it is quite essential to translate them in terms of metrics and numbers as much as possible. Clear and straightforward metrics should be put in place as they help enterprises to measure their improvements, giving them a proper idea of where they stand, and providing direction for the project. It is also essential for enterprises to set up a target of what they are going to achieve as part of the test process assessment and based on this the entire process can be divided into multiple phases.

Step 2: Assessing current test processes by performing reviews and conducting interviews:

Based on the test organization's business goal and the assessment goal, the method of performing assessment is decided. There are two methods of performing the test process assessment:

1. A formal test process assessment which is more methodical, more structured, and often more time consuming.
2. An informal, less structured test process assessment based on the specific needs, which consumes less time.

In an organization testing typically doesn't exist in isolation. It is aligned with several business teams, development teams, vendors, test environment teams etc. The major stakeholders who are mostly impacted or influenced by the test teams in an organization should be identified. In parallel, some artifacts from the existing projects should be collected and an offline review

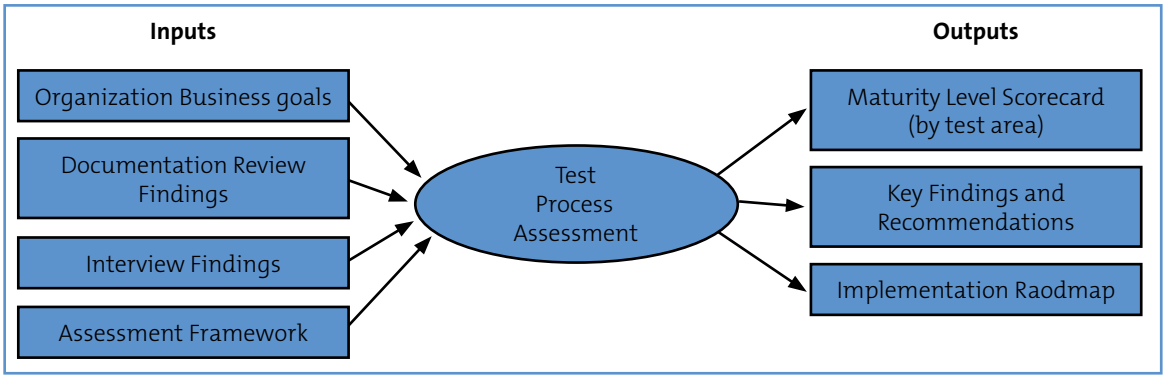


Figure 2: Input and output entities of assessment

of these should be done. A questionnaire based on the business goals should be prepared and given to the stakeholders. Once the responses from the stakeholders are received, those should be synthesized with the collected artifacts of the previous projects.

The figure 2 gives a pictorial representation of input and output entities of a test process assessment.

Once the necessary knowledge is gained from the previous artifacts and the responses from questionnaires, a list of questions should be prepared for conducting interviews with each of the stakeholders. Interviews should be established with each of the stakeholders, and focus should rest on identifying some of the conflicts in terms of what's really happening in an organization versus what the stakeholders are saying. More exploratory questions based on test organization, organizational test strategy, organizational test process, project test process and methodology, test and environment management and testing review and optimization should be posed in the interviews, and more inputs must be gathered on the current testing processes in place. The interview questions should be customized based on role of the stakeholder in the organization.

Step 3: Reporting assessment findings and establishing roadmap for implementation:

Data gathered from different areas and from different stakeholders gives a clear understanding of the current processes of the organization, its capabilities, and what they want to achieve. The necessary improvements based on the requirements of the organization can be established. For example, there might be few processes that are not up to the mark, but they can be ignored as they do not have any impact on the goals of the organization. Examining all these factors helps in recommending a set of necessary improvements and metrics that can be immediately implemented within several project teams in an organization.

The recommended improvements can further be classified for short-term, medium-term and long-term implementation.

Step 4: Implementing recommended improvements:

Once the necessary improvements have been established based on the business as well as testing goals of an organization, the next step involves assigning priority to the suggested improvements to clarify what they want to improve on immediately to gain quick wins, and what they want to address over a period of time, and also how they would like to implement those improvements,. Priority can be assigned to the improvements based on several parameters, such as time required to implement the respective improvements, investment required in terms of people, hardware, and software, the level of difficulty in implementing the change, the impact the respective improvement is going to have on the testing processes etc. Based on these parameters, a prioritized list of recommendations should be prepared and submitted to the sponsor for approval.

Typically, the project is done in short term (0-3 months), medium term (4-6months), and long term (up to 1 year). A plan should be put in place and then a consensus should be established on what the organization wants to do and how they want to go about it.

Step 5: Measuring post-implementation effectiveness:

Once the improvements are implemented, the team gathers feedback from various stakeholders at defined intervals. The feedback gathered is analyzed to understand the effectiveness of the improvements and identify any further changes that need to be made.

A detailed report on the benefits achieved through implementation of improvements is presented to the sponsor and to all other stakeholders involved.

What best practices to use in implementing Test Process Improvement?

The test process improvement should be done in an incremental way, as instant changeover by adopting the Big Bang approach would not be the right way to implement it. Some guidelines for a successful test process improvement are: try not to change too much but implement a few significant changes that take an organization forward, analyzing the priority areas and working on them, and making changes to the areas where impact can be very high. At first, the process changes should be implemented in one pilot project or one unit that is in its earlier stage and bears a low amount of risk. The testing experience gained and the testing process implemented from the pilot project should be used as a case study while implementing test process improvement for other projects. So, test process improvement has to be a mix of enforcement backed by proven results while implementing in some other areas.

The improvements should be thoroughly implemented according to a plan and regularly monitored to identify whether the improvements made are meeting the requirements or business goals. Test Process Improvement should be done by taking into consideration various recognized industry test maturity models, such as the Capability Maturity Model Integration (CMMi) and Test Maturity Model Integration (TMMi) as the reference, so that the current scenario of the test process can be analyzed well.

The CMMi identifies a core set of software engineering process

areas, such as requirements development, requirements management, technical solution, product integration, verification, and validation, while also covering process areas, such as process management, project management and support. The TMMi contains stages or levels through which an organization passes as its testing process evolves from one that is ad-hoc and unmanaged, to one that is managed, defined, measured, and optimized. Achieving each stage ensures that an adequate improvement has been laid as a foundation for the next stage. The five levels in the TMMi Model such as, Initial, Managed, Defined, Management and Measurement, and Optimization, prescribe a maturity hierarchy and an evolutionary path to test process improvement.

Benchmarking the test processes with industry recognized test maturity models helps organizations to assess the quality of their test processes, assists in identifying the necessary changes, and gives them a direction for the improvement activities, ultimately leading to the success of the project.

What are the benefits when you drive assessment & improvement upfron?

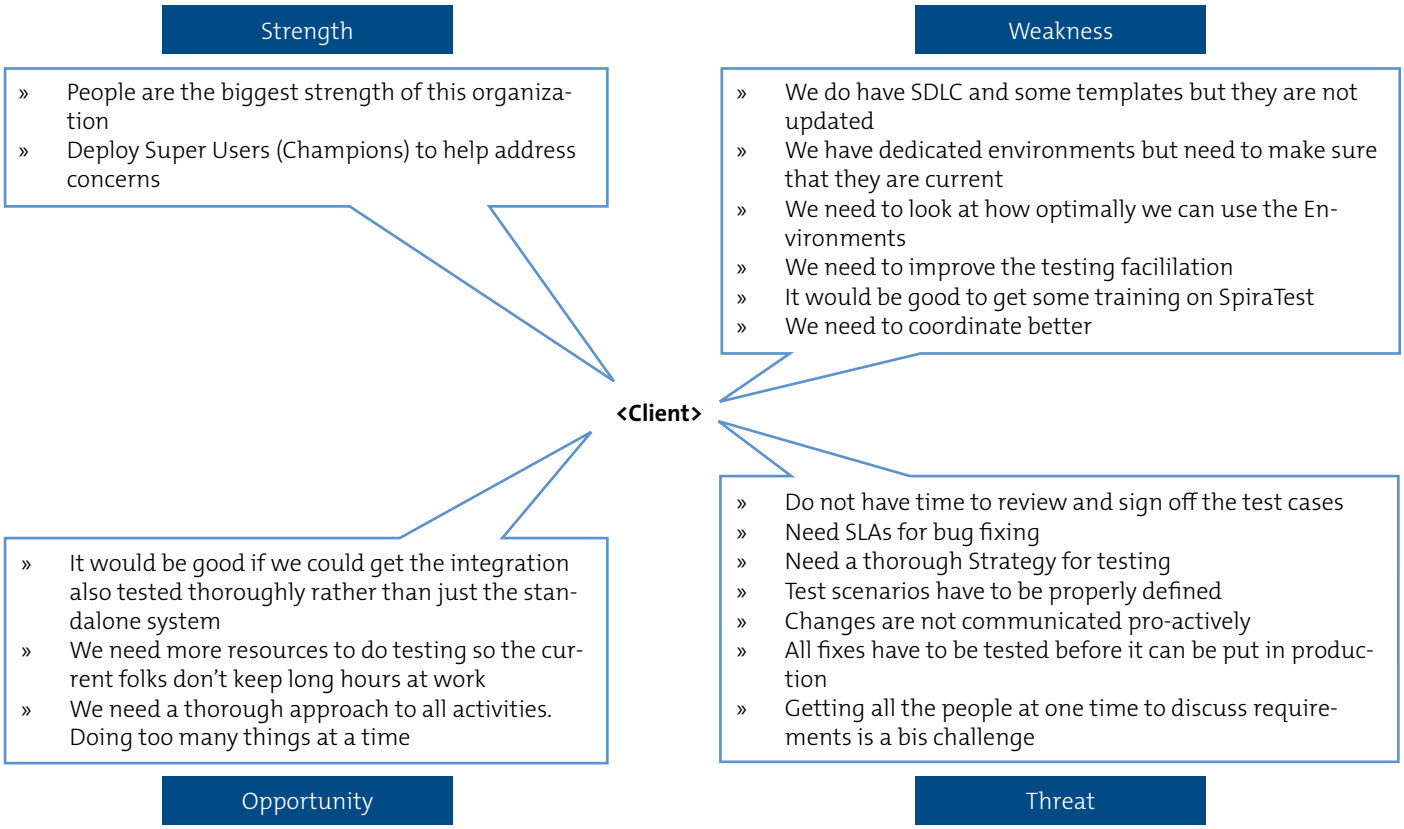
Test Process Improvement helps enterprises to achieve the desired test maturity levels and more streamlined and efficient processes. It enables organizations to optimize their testing performance and lead time of the test process, ensuring overall quality of the IT systems. It involves several other sub-elements such as defect management, testing techniques, test tools, test automation, test strategy, test environment, test coverage, test process innovations, test vision and goals, roles and responsibilities etc. It is therefore essential to identify the focus areas of enterprises for improvement, since all areas may not be at the same maturity level.

Various other benefits include, but are not limited to the following:

- A clear assessment of an organization's current strengths and weaknesses in the testing as well as the software development life cycle processes compared to industry best practices
- A clear roadmap indicating the priorities of recommended improvements
- Establish a baseline upon which future improvements can be made and measured
- Set up realistic goals and achieve them based on certain metrics
- Proven frameworks to meet various regulatory mandates
- A well established test process results in good ROI due to increased quality and alignment with business goals
- A framework to ensure that the estimated ROI from test improvements is being monitored and achieved
- More efficient and effective business operations
- Enhanced control over the testing process as well as the associated costs
- Reduced down time
- Reduced errors
- Lower production maintenance overheads
- Long-term reduction in the cost of testing
- Long-term reduction in the length of the test process
- Increased efficiency with respect to the development and testing life cycle and supporting process
- Increased quality of code into production and code into each of the test phases

The overall aim of the Test Process Assessment and Improvement is to ensure long-lasting improvements that optimize a company's return on investment and add value to the business.

Sample representation:



Segregation of assessment findings by SWOT

Recommendations by test area – Test metrics:

Strengths			
• Reports are generated from Quality Center to inform the progress of testing to stakeholders • Quality goals are documented in test strategy documented, however they are not monitored • SLA's for defect resolution exists in test strategy document, however no evidence is found for monitoring • Investment made in using quality center as a test management tool			
Areas of improvement	**Recommendation**	**Benefits**	**Priority**
Measurable quality goals are not set for projects	Quality goals have to be set right in the beginning of the project during planning phase with inputs from client and other key stakeholders	Helps in tracking the project progress against the set quality goals	High
No defined measurement framework in place	Comprehensive metrics framework to be in place	To understand the current quality levels of the applications and to drive continuous improvements	High
No Metrics exist to measure (1) quality of the appliocations (2) quality of testing (3) progress of testing (4) Test team competency (5) over all project quality goals	Metrics such as Defect severity index, Test case authoring and execution productivity, review effectiveness, test effectiveness should be defined, captured and analyzed for improvements	Helps in making Go-No Go decisions Helps improve project processes Increased customer satisfaction by ensuring higher test effectiveness	Medium
Process capability base-line does not exist at the test organization level	Establish process capability base-line at the test organization level	Improves over all test maturity of the organization To set quality goals for projects	Medium

> biography

Kalyana Rao Konda
is the Vice President, Delivery at AppLabs, the world's largest software testing and quality management company. He has been with AppLabs since early 2005. Kalyan has over 12 years of IT experience in diverse areas, including global delivery management, talent development, test management and non-functional testing. After securing a B.Tech in Electronics and Communication engineering from Pondicherry University, he started his career in the field of software testing, and soon built an impeccable track record of managing large scale complex testing projects across a wide variety of organizations. Prior to joining AppLabs, Kalyan worked with Virtusa and BaaN. Kalyan has also published several papers and presented at international testing conferences.

How Utility is assured success of its AMI implementation program?

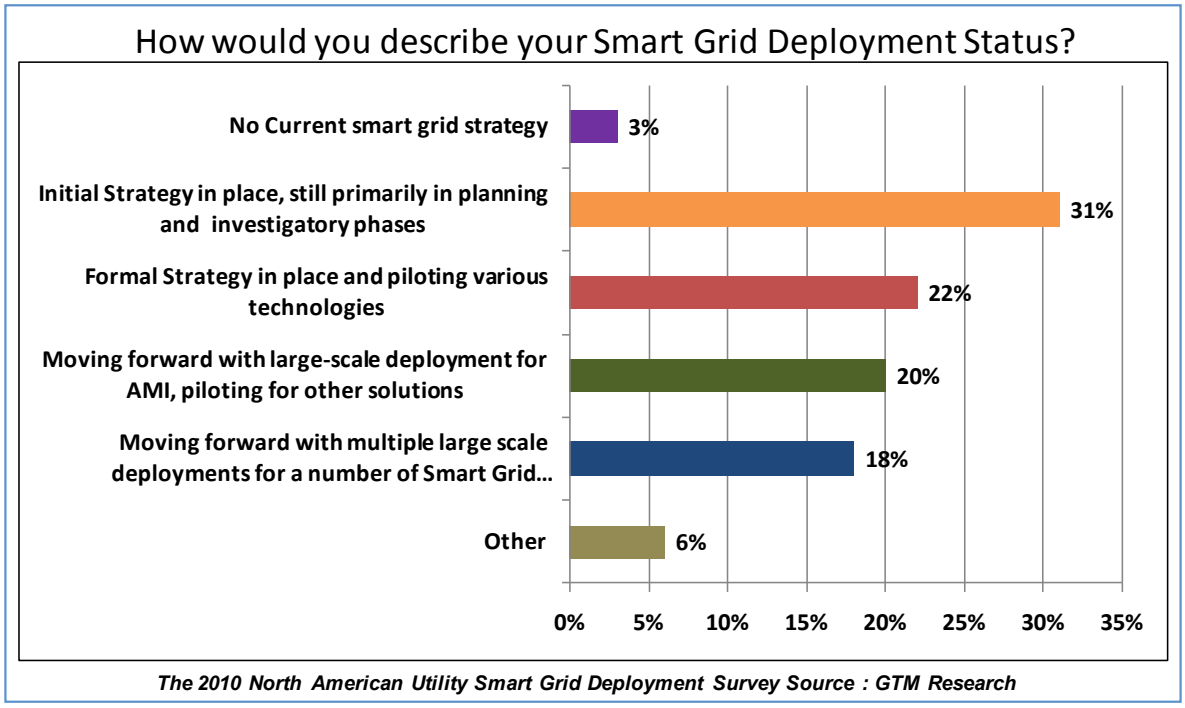
by Priya Rajavaram & Babulal Prasath

Domain-specific testing is becoming more and more relevant in today's world where the software time to market is a key factor in its success. Before a software product is launched, the next version is already available in the market. With a time-constrained application development cycle, it becomes imperative that System Integration testing focuses on business relevant testing and ensures optimum coverage and at the same time high quality software. Today software testing services are increasingly focused on having in-house SMEs and domain experts who know the business and can work in a dynamic software development cycle to meet the risks arising from lean requirements and design elaboration and documentation, shortened review cycles and reduced SME bandwidth availability. This is especially challenging if the domain brings with it technology challenges. Smart Meter or Advanced Metering Infrastructure is the buzz word in the utility domain.

Introduction to Advanced Metering Infrastructure

In the utility sector, advanced metering infrastructure (AMI) is burgeoning as a method for providing measurement of energy for the 21st century. Within the last year, more than 25 million meters worth of AMI business in North America, valued at more than $2 billion, has been put out for bid. This compares to the installed base of roughly 80 million meters operating under automated meter reading (AMR), a precursor to AMI. [1]

Governmental projections show that by the year 2025 there will be 2 billion more people inhabiting our planet, consuming more energy. Energy demand will rise by 54 percent. Electricity demand in the U.S. is expected to grow by 141,000 megawatts in the next decade, while only 57,000 megawatts of new resources have been identified. Environmental, legal and social pressures already constrain where and how fuels are obtained, generation plants are built and transmission lines are located. Without action, any energy shortage will become markedly worse. [2]



The 2010 North American Utility Smart Grid Deployment Survey Source: GTM Research

Utilities all over the world are either implementing or have plans to transform their business by embracing Smart Metering or AMI. The core of the Utility business is its Meters. Meters are the source of all measurement and basis for any intelligent analysis. "Advanced metering is a metering system that records customer consumption [and possibly other parameters] hourly or more frequently and that provides for daily or more frequent transmittal of measurements over a communication network to a central collection point."[1].

AMI also provides more flexibility to the customer by providing them with the information of volume of their utility usage at different times during the day. Key drivers for AMI are:

- AMI (Advanced Metering Infrastructure) is one of the key focuses in the Utility Industry all over the world today. It is defined as a system that will be capable of collecting data from Electric/Water/Gas Meter remotely at a defined interval.
- Governments all over the world have directed and are encouraging utility companies to implement AMI.
- AMI automates a lot of current manual operations - like meter data collection, TURN On, Turn Off, Service Transfer, Outage Management , Tampering & Theft detection, etc. This provides the Utility sector a huge return on investment by adopting lean operations and drastically cutting down costs.
- Utility companies also see huge spurt on their customer experience by enabling them with hassle- free self-service features and providing the customer with usage history and trend analysis. Value added services like analysis and best practices on reducing usage and promoting savings can be offered. The foremost use of the AMI technology is that it can be used to promote concepts for saving energy. This is achieved by providing incentives or lower rates when a customer uses less energy (say less use of A/C) during the hours of peak demand. This can prove to be huge saving of non-renewable energy sources. This is the primary reason why governments are ready to invest in the AMI projects.

Challenges in AMI implementations

AMI implementations are massive transformation programs which typically require a timeframe of between 3-5 years for a full-scale rollout depending on the customer base of the utility. The following are key challenges in any AMI implementation:

Cost

Deploying smart meters requires huge capital investment and hence it is crucial that the utility company is confident of successful implementation.

Data Privacy

There are many concerns related to privacy of consumption data being raised as Smart Meters are installed at more and more locations. The meters' data can be mined to reveal details about customers' habits like when they eat, how much television they watch and what time they go to sleep. The retention and storage of this data make it vulnerable to security breaches as well as government access.

Technology Transformation

The imminent rollout of smart metering in the short to mid-term future will requires vital investments in information technology for setting up turnkey projects for smart metering network and application infrastructure.

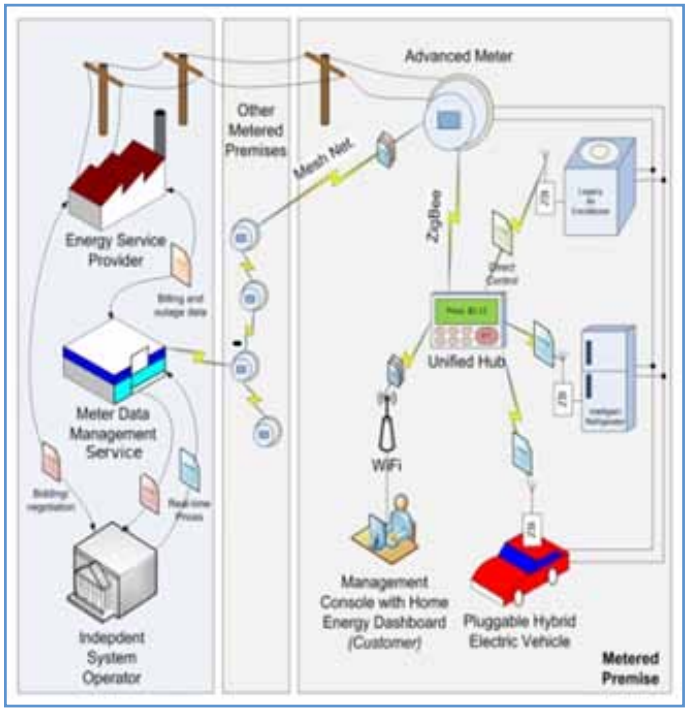
It will be necessary to build up components like

- Mass deployment of smart meters
- Meter reading databases and applications
- Data warehouses and analytic tools & data migration programs
- Customer interfacing applications and portals
- Integration with existing systems

These components and underlying infrastructure will be critical success factors for generating maximum success from smart meter rollout programs.

AMI implementation brings with it a transformation in the entire utility business process, and impacts the entire span of utility operation & stakeholders. Hence validation of AMI implementation becomes a very crucial task. It becomes imperative that the utility validation partner in these kinds of programs understands the business domain, and ensures that the application not just conforms to the business requirements but is also able to meet other criteria (e.g., performance, data quality & security, application security).

Advanced Metering Infrastructure Landscape



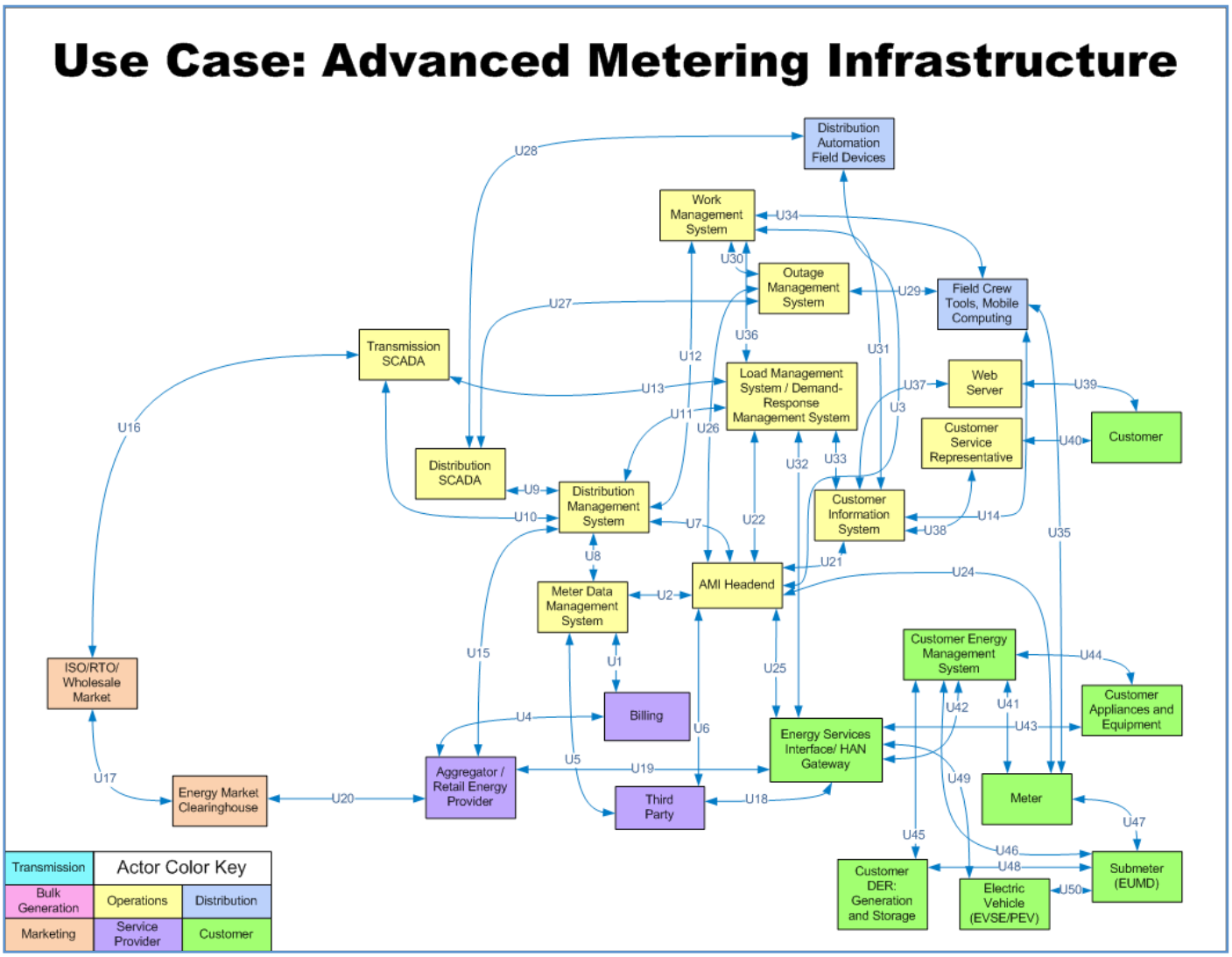
Source: http://seclab.uiuc.edu/web/component/content/article/37-education/81-ami-security.html

Key validation challenges and how domain knowledge plays a vital role

AMI implementation programs typically are huge programs with multiple programs running in parallel to achieve the overall objective. The key challenges are:

Dynamically changing requirements

Testing based on Business Requirements Document, or on design documents becomes irrelevant when it comes to AMI validation. As is the case with any large transformation program, the carefully crafted requirements document, which is signed off and approved after much deliberation, becomes obsolete from the day the project picks up steam. The role of the decision/consensus document or the deviations approval document come into the picture, which is where these decisions are documented and



Source: http://collaborate.nist.gov/twiki-sggrid/bin/view/SmartGrid/CsCTGArchi-Usecase-AMI-Diagram

implementation go-ahead is decided. In these situations, where there is no detailed documentation for these quick decisions taken by the team, it is imperative that the QA organization is agile enough to assess impacts both on the technology front and underlying implications to the business and the impact on downstream functionality.

Deployment of Smart Meters

The deployment of smart meters in such a huge volume demands a very highly organized system with properly managed inventory control. If this mass deployment is required to be carried out by a third party vendor, the challenges increase as co-ordination and synchronization between the inherent systems can be very complex. A mass deployment project also requires a lot of investment in training of the meter operators, inventory management and safety

Testing of a complex billing system post AMI implementation

The billing system by itself is of a complex architecture with its various types of service plans, tiers of use and categories of customers. Along with the introduction of interval data, that can be as frequent as every 15 minutes, this doubles its complexity.

- Calculation of the billing determinants during a previously announced rebate time
- Calculation of the billing determinants during a previously announced critical peak time
- Calculation of billing determinants for on-demand billing requests which might be for a shorter or longer period than the regular monthly bill.
- Adjustments of bills based on the previous billing period's balances/overcharges
- Failover case testing of Smart Meters

These are the main factors which added to the complexity of testing.

Testing of new product implementation- MDMS

MDMS stands for Meter Data Management System which is responsible for storing and providing the interval data for each of the Smart Meter connected to its network. The MDMS product has to be crafted to fit the existing billing system and still has to be robust to handle the interval data at each meter/account level. In other words, the new product has to be customized to fit the existing billing framework and has to extract and provide the billing determinants in such a way that the billing system can readily consume it to produce the bill. This involved a lot of challenges from the design phase through to the end of the testing phase. A lot of configuration changes had to be done to make the interface work. The lessons learned taken from the product side from such a big implementation should be standardized and form the baseline for future AMI implementations.

Thursday June 23rd 2011 - Conference Center Figi - Zeist

test automation day

www.testautomationday.nl

Optimize the profits of the next generation Test Tools

It is a great pleasure to welcome you to Test Automation Day 2011 in the Netherlands!

The conference committee has organized an exciting collection of keynote sessions, business cases and workshops presented by national & international thought leaders and experts.

Keynote speakers are

Bob van de Burgt - *Chairman* -, Testadvisor, former president of TestNet, Programme Chair EuroSTAR 2008 and author.

Scott Barber, CTO PerfTestPlus, Co-Founder of Workshop "On Performance and Reliability".

Mark Fewster, independent consultant specialising in software testing, Co-Author of the book "Software, Test Automation".

Martin Gijsen, Test Automation Architect & Coach, deAnalist.nl.

Arie van Deursen, Professor in Software Engineering, Delft University of Technology.

Register with the special discountcode!

Register now and explore the next generation test tools on Test Automation Day 2011! **Participation fee for readers of Testing Experience is only €195,-!**

Registration: www.testautomationday.nl

Discountcode: TAD2011_TEXP

Participation fee only **€195,-** for readers of Testing Experience

See you on June 23rd!

Congresorganisatie ckcseminars

founding partner

partners

sponsors

exhibitors

with contribution of

JUNE 16TH 2011 - WORLD TRADE CENTER - ROTTERDAM, NETHERLANDS

Innovate IT 2011: Innovation through Cloud Applications

Visit the Innovate IT Conference on June 16th and **receive a € 100,- discount!** Including leading experts such as **Peter van Eijk** (Computable), **Matt Wood** (Amazon Web Services) and **Hans Appel** (Hanze University, Groningen).

Registration: www.innovate-it-conference.com. Discount code: 'TE-100'. Admission is FREE for every 2nd person of the same organization.

More information: **www.innovate-it-conference.com**

Data Migration and Data warehouse testing

Smart Meters introduce interval meter readings, which necessitates that this voluminous data is handled carefully and utility IT systems have the ability to store the data and save historic data. Data migration and data warehouse testing are key projects to a successful implementation of the AMI initiative. However, functional knowledge is required as this data (meter interval data, usage, billing determinants, etc.) is critical and the source of all analytics.

Validating Web Portal upgrades

One of the main drivers of AMI implementation is to enhance the customer experience. Upgrading the website to ensure that a customer switching to Smart Meters can experience the benefits of the initiative. The key validation areas are:

- Customer - My Account
- Understanding Usage & Bills
- Rate plans
- Payments & credits
- Reports & analytics
- New initiatives like energy audit, demand response & demand-side management.

Integration Challenges

AMI implementation is a vast transformation program running over a span of 3-7 years. It is a huge project and needs to be implemented in phases. It is very important that the AMI program is broken down and taken up one after another. However, this brings with it some integration challenges. While new systems are being implemented (like MDMS), legacy applications (CIS), datawarehouse and portal upgrades take place. It becomes a challenge to ensure integration points are not failing. Interdependencies of various systems cannot be identified just based on the requirements document, and a tester should be equipped with domain knowledge. This will ensure that the integration points are well defined and all relevant data is flowing to & fro from different systems to ensure the end-to-end business scenarios. A domain expert can also highlight the need for availability of upstream or downstream functionality and ensure that the project timelines are synchronized and kept.

Test Data Management

Planning, creation & managing test data at an overall program level is a very key task. Most test scenarios cut across various applications and unless test data is planned upfront, testing is bound to end up as a total failure. The typical cliché of Garbage In Garbage out holds true without proper test data planning. Much effort is required in coordinating with different departments: from CRM to get the right customer accounts for testing different scenarios, to Billing to schedule bills, to Payment & Credits to update payment/defaults based on the scenario to be tested. Utmost care is to be taken to ensure the test data is valid at the time of testing.

Understanding what batch jobs are required to update the status of scenarios and which department has to be notified about the same saves a lot of last minutes glitches.

Conclusion

Domain testing is an imperative in a complex domain intensive area like Advanced Metering Infrastructure. Requirements based testing does not help, as there are many unwritten rules which can be tested thoroughly only by someone who understands the business domain. The systems need to be tested for end-to-end scenarios which cut across the entire Utility application landscape. For such challenging systems, understanding the domain and validating end user scenarios need intensive domain knowledge coupled with technology skills.

References

[1] http://www.nextgenpe.com/article/The-Critical-Role-of-Advanced-Metering-Infrastructure-in-a-World-Demanding-More-Energy/

[2] http://www.itron.com/pages/solutions_detail.asp?id=itr_016422.xml

[3] http://collaborate.nist.gov/twiki-sggrid/bin/view/SmartGrid/CsCTGArchi-Usecase-AMI-Diagram

[4] http://seclab.uiuc.edu/web/component/content/article/37-education/81-ami-security.html

> biography

Priya Rajavaram
is a Test Manager at Infosys. She is currently leading IP solution development in Utility (Advanced Metering Infrastructure) space and manages a leading US Utility's testing for its Transformation program.
During her stint of 10.5 years she has handled various engagements. Some of her responsibilities in past are listed below:

- *Test manager & process consultant for one of the world's largest integrated energy companies.*
- *Test Manager & Sub Vendor QA Manager for a leading European Airlines Alliance. Handled multiple stakeholders' coordination - IT sub vendor and all 19 airline alliances.*
- *PMO & QA Manager for UK's largest airline.*
- *PMO for a Global GDS provider.*

Babulal Prasath
Babulalprasath has been working in a leading US Utility's AMI implementation program as Test Analyst. He has been involved in all phases of testing of the AMI implementation project starting from AMI implementation vendor selection by product testing, pilot roll out of smart meters, mass roll out, MDMS implementation and integration with legacy system and currently is involved in future implementations of AMI program. He has a very good knowledge on CIS business domain and AMI application architecture as a whole. He has been the primary member and has played a pivotal role in mentoring a big team at onsite. He also has worked on testing projects of banking domain with one of Infosys vital clients.

Round a Ring of Processes, Pots Full of Models...

by Sharon Wei Minn Cheong

Software models, test processes ... it's like a merry-go-round. Trying to figure out which model goes with which process, or the other way round is like playing hide-and-seek. Often when software models and test processes are identified and brought together, Test Managers will have a field day figuring out how to improvise the test methodology into the project methodology. That's when the improvising game begins. How can Test Managers improve test processes to have them more maintainable, flexible, and faster in delivery time, without undermining the quality of the product in question? Why is improving the test process so important in software delivery, and how feasible is the improvement process for risk-based testing? What test model fits best into the current software model that is implemented? This article will first define the basic understanding of what test process can be beneficial to the project team. Finally, the heart of the article will describe various project models that can be integrated with some examples of test process improvements.

What is a test process?

A test process is like a skeleton to a house. Before a house is built, raw materials are acquired, architecture designs are studied and calculative analysis is performed. As such, if we think of a test process in a project as one of the houses to be built for the test team together with the software development team as well as the analyst team, we will have a neighborhood established. A well-defined test process will instill proper communication and collaboration among our neighbors, providing a smoothly delivery of the product to the clients. Simultaneously, it serves as a back bone and check point for every phase in the project, from start to finish, and slowly integrates with the maintenance team for further regression testing to be performed later down the road. With a structure in place, a more solid foundation is built for the purpose of testing software quality, ensuring that delivered products abide, and are in accordance with the stipulated requirements. The structure subsequently allows Test Managers to back-track defined tasks to ensure that all deliverables listed are provided in a timely fashion.

Why is improving the test process crucial to software development and to the project team?

Continuously improving the test process fits entirely with how human beings are. We learn every day to make things better for ourselves and improve our knowledge for better communication and purpose in life. The moment we stop advancing forward and improving our knowledge base, will be the moment where we become obsolete.

Hence, the following benefits are highlighted in why improving the test process is crucial to the project team.

- *Improve software quality*

Most of the test process improvement efforts are focused mainly on the quality of testing. Subsequently, the software product is affected by the improvement effort undertaken, which in turn also provides qualitative software.

- *Cost effectiveness*

Test process improvement can also improve the cost effectiveness of a test process, by improving efficiency and effective delivery of software, which saves money time and effort.

- *Timely delivery*

If a test process is not improved in the long run, plans tend to slip, which causes delays to software delivery. As such, by improving the test process, delays can be avoided and consequently produce more feasible and reliable plans to be implemented.

- *Team assimilation and collaboration*

Test process improvement can also provide team assimilation in a project team, instilling team work and collaboration among all sectors of the project. The collaboration effort will get everyone on the same page, geared towards the same objective, which will result in the development of better software quality products delivered on time.

When to perform test process improvement?

- *During the start of the project*

At the start of the project, it is useful to analyze and come up with a good test process before it is even written. It is always good to start with a strong foundation and have a holistic approach, en-

suring all risk areas are covered, rather than to have a poor process as a base, which will require more time to improve along the project life cycle.

- *Throughout the project life cycle*

Throughout the project life cycle, changes will occur as the integration of the software development process as well as the test process together with the project methodology are put to test. During this phase, additional suggestions to provide room for improvement are executed in order to let everyone from each team see eye to eye. This is beneficial when the process is still undergoing relevant usage to the project team, and therefore, improving it will benefit the collaboration between teams for better communication and better quality output.

- *At the end of the project*

Normally at the end of any project, a "post mortem" exercise is conducted to identify the lessons learned from the various activities undertaken throughout the project life cycle. The reason is to ensure that we learn from mistakes and that process improvement can be fine-tuned ensuring that the same errors or mistakes will not be repeated in other projects or in the extension of the existing project. Most of the information gathered will be documented as process improvement actions.

- *During maintenance phase*

During the maintenance phase, often enough, test processes are ignored or forgotten. The integration of customer care or follow up is not even thought of, during or throughout the project life cycle. All teams are absorbed in delivery time and process improvement, so that when a product is delivered, the support team takes over with a different process. However, when change requests or bugs are identified after delivery, regression suites and the process to develop and execute those tests are needed. As such, with a test process in place, maintenance testing is included as part of the project life cycle.

How to improve the test process?

Step 1: Identify and Initiate

The first step is to identify what test process needs to be used for which model. This will start the basis of software testing implementation. Of course, software processes, requirements as well as development and test plans will have to be in place beforehand. This will provide the objective and scope coverage of the process improvement to be defined more accurately.

Step 2: Measuring and Baselining

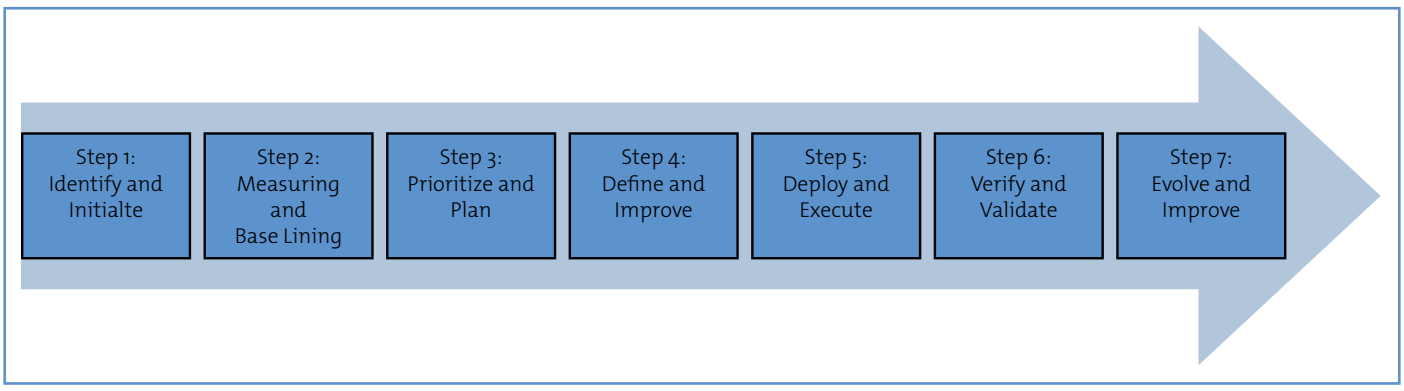
Test criteria and agreement in understanding the same commitment or objective will be defined in this step. Agreed requirements will be the baseline, and measurement metrics are identified using previous analysis collected from previous projects to improve matrices that are already in place, or new metrics will be defined instead to cater for the objective stated. With measurements such as the defect or risk metrics, the project team will know where they are in terms of quality and delivery time.

Step 3: Prioritize and Plan

Based on the identified process and measurements provided from steps 1 and 2, prioritization and planning begins. Several factors should be taken into consideration during the prioritization and planning stage, such as return on investment, risks and organizational strategy, and these must be in line with the identified objectives. The question of what, how, when and who will be addressed at this stage as well as the output of such questions will be combined in the project planning document.

Step 4: Define and Improve (Define and Redefine)

At this stage, the implementation strategy begins. Implementation solutions will be provided and executed. Any further improvement required will be redefined and restructured for further enhancements. Practicality and objective output is the goal



to achieve at this stage. Additionally, the involvement of the people who will be using the process is addressed in this step.

Step 5: Deploy and Execute (Operate)

When the test process has been defined, the actual deployment and execution is started. This is of course easier said than done. Some considerations to take into account:

- Ensure a small scale of deployment is performed. This can also be called the pilot phase, in which small changes are implemented to ensure that no conflicts occur between the test process improvement models and the software development model.
- Ensure that the necessary training is provided for all affected people in using the new process.
- Ensure that the required support and hand-holding presence is provided for using of the new process.

Step 6: Verify and Validate

When the test process is fully implemented and integrated within the project model, the verification steps begin to ensure that what has been implemented is working accordingly. This is just like comparing actual results to the expected results; the planned benefits and success criteria have to be compared to the actual situation. Artifacts from various input sources, such as review process, test design and execution results, are taken into consideration for the validation activity.

Step 7: Evolve and Improve

The re-evaluation of the test process will need to be evolved and improved along the way. What may work at the beginning, may not work during the process life cycle. Based on the gathered analysis output and data, improvement will be a continuous activity.

Integrating Test Processes with project models?

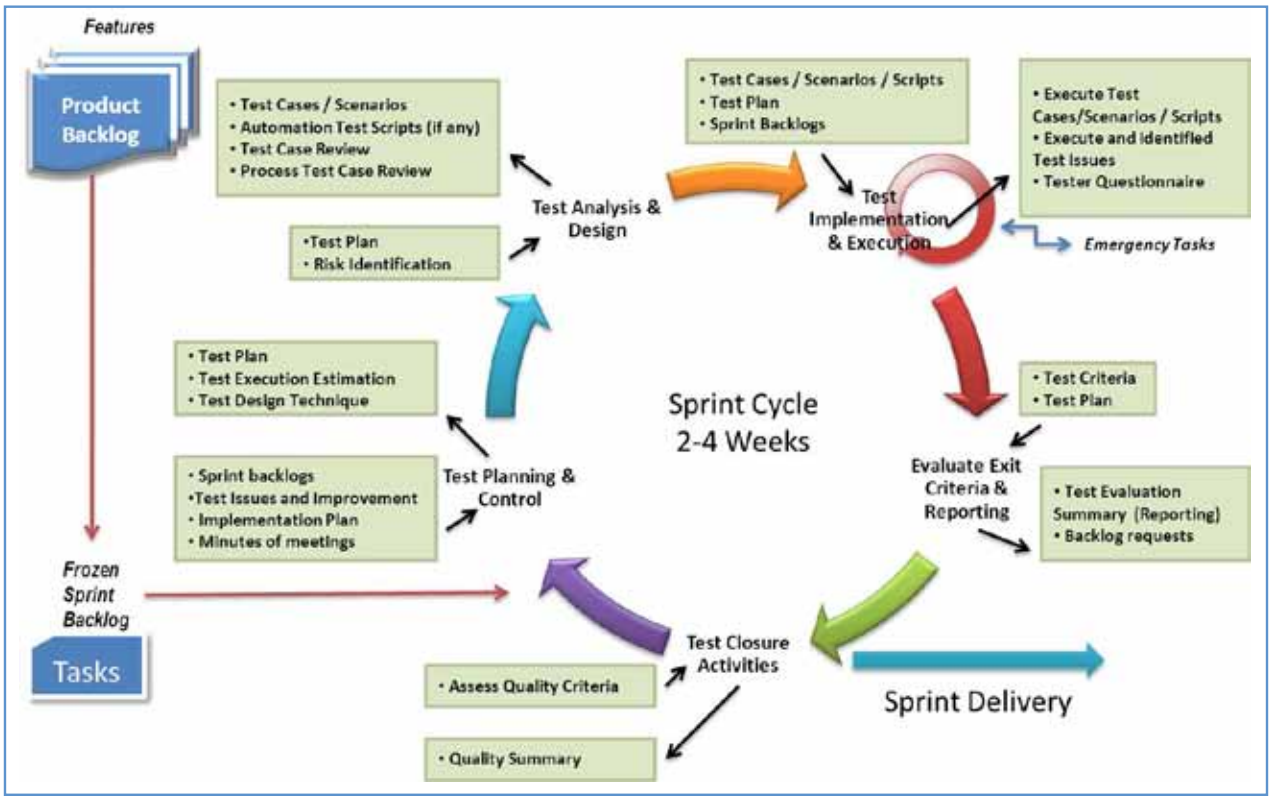
Flexibility in improving test processes is always a challenge in any project or mission. Depending on the test process itself, the basic foundation of test planning, test design, execution and implementation as well as reporting, is a must-have in any testing life cycle. However, the question of how and in which stage of a software model to integrate these processes without too much risk is the challenge. RUP and Agile /SCRUM are the most practiced software development life cycle models implemented. Additionally, Test Process Improvement (TPI), Critical Testing Process (CTP) and Test Maturity Model Integration (TMMi), to name a few, are process models which are still emerging. Integrating both test models and software models to communicate with one another, while instilling team work and communication, will no doubt increase productivity and efficiency in software delivery. The success of such integration is like a marriage made to improve over time.

Selecting the correct model for a project can sometimes be challenging. The following similarities are identified with various test processes, whereby naming conventions and implementation

<table>
<tr><th>Generic Test Process</th><th>TPI (Cornerstones)</th><th>TMMi</th><th>CTP</th></tr>
<tr><td>Test Planning & Control</td><td rowspan="4">Life Cycle
Key areas:
1. Test Strategy
2. Life Cycle Model
3. Moment of Involvement
Techniques
Key areas:
1. Estimation and Planning
2. Test Specification and Techniques
3. Static Test Techniques
4. Metrics
Infrastructure
Key areas:
1. Test tools
2. Test environment
3. Office environment
Organization
Key areas:
1. Commitment and motivation
2. Test functions and training
3. Scope of methodology
4. Communication
5. Reporting
6. Defect management
7. Test ware management
Test process management
8. Test process management</td><td rowspan="2">Level 1: Initiate
• Define best test practices
Level 2: Definition
• Test Techniques and Methods
• Test Planning
• Test Policy and goals
• Test environment
Level 3: Integration
• Control and monitor
• Test life cycle and integration
• Test training program
• Software test organization</td><td>Phase 1: Plan
• Understand Context
• Prioritize Risks
• Estimate Budge
• Make Plan
Phase 2: Prepare
• Make Team
• Implement test system</td></tr>
<tr><td>Test Analysis & Design</td><td rowspan="2">Phase 3: Perform
• Manage test releases
• Execute Test cases</td></tr>
<tr><td>Test Implementation & Execution</td></tr>
<tr><td>Evaluate Exit Criteria & Reporting</td><td>Level 4: Management & Measurement
• Software quality evaluation
• Test measurement program
• Peer reviews</td><td rowspan="2">Phase 4: Perfect
• Writing bug reports
• Reports the test results
• Manage change
• Perfect the testing process</td></tr>
<tr><td>Test Closure Activities</td><td>(ALL)
Key areas:
1. Evaluation (reviews)
2. Low Level testing</td><td>Level 5: Optimization
• Test process optimization
• Quality control
• Defect Prevention</td></tr>
</table>

phases may differ slightly, but the similarity in test activities to be performed is acutely similar. As you can see from the table, the Generic Test Process consists of a sequential life cycle to be followed. Compared with TPI, TMMi and CTP, the overlapping or similarity in process is parallel but only at a different state of the Generic Test Process.

Once the identification of the test process is selected, the integration process with project models begins. Integrating test processes with project models is often like a jigsaw puzzle, constructing it to fit. An example is shown with the generic test process and the Test Maturity Model integration (TMMi) process into an Agile/SCRUM software model.



A generic test process consisting of standard test activities integrated with the Agile/SCRUM software model. Similarly, the same concept of integrating the Agile/SCRUM model with a TMMi test process is also applicable.

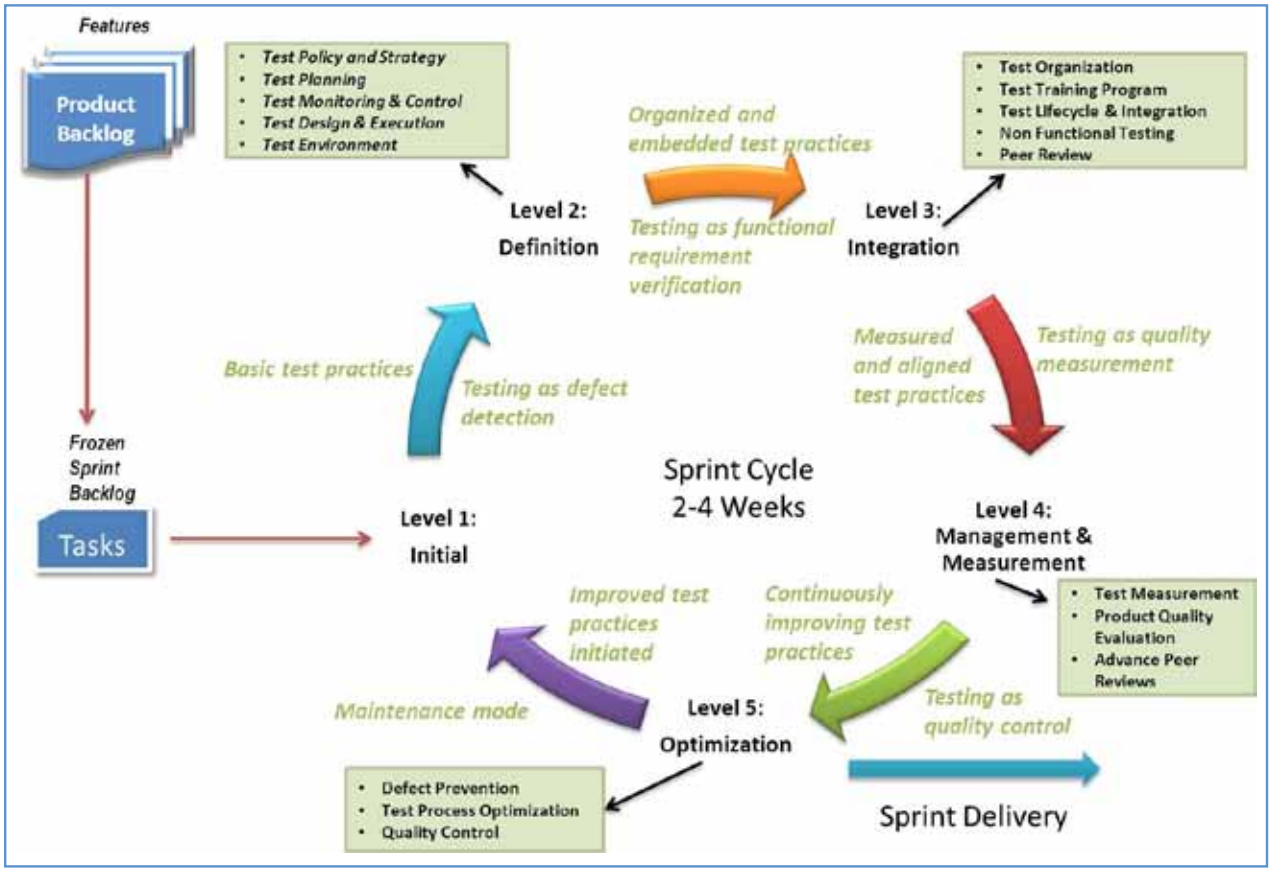


Figure 1: Integrating Agile/SCRUM software model as an example to both Generic Test Process and TMMi

Another example would be various test processes with the Rational Unified Process (RUP) model.

Conclusion

As presented, improving test processes in any project model is feasible and attainable. Various processes presented can also be incorporated for efficiency and to ensure quality software is still preserved. Generally, the objective of the test process life cycle is the same. Any test process improvement that is in place will integrate successfully with any project model as long as room for improvement exists.

References

Software Testing Practice: Test Management: A Study Guide for the Certified Tester Exam ISTQB Advanced Level (Paperback) - Andreas Spillner (Author), Tilo Linz (Author), Thomas Rossner (Author), Mario Winter (Author)

> biography

Sharon Wei Minn Cheong *has over 10 years of international experience in the software development industry. She has experience in software development, project management and quality improvement processes in Asia, Europe and the Americas. Sharon is responsible for process improvement and function as a Test Manager for Trasys Greece. She has a bachelor degree in Business Information Technology and a Master in Business Administration from Cardiff Business School, Wales/UK. Sharon is an ISTQB certified test professional and has gained her experience through orchestrating test management processes in the aviation, pharmaceutical and chemical industries. Originating from a software development background, Sharon sees processes in a more holistic approach and positively integrates both technical and business needs.*

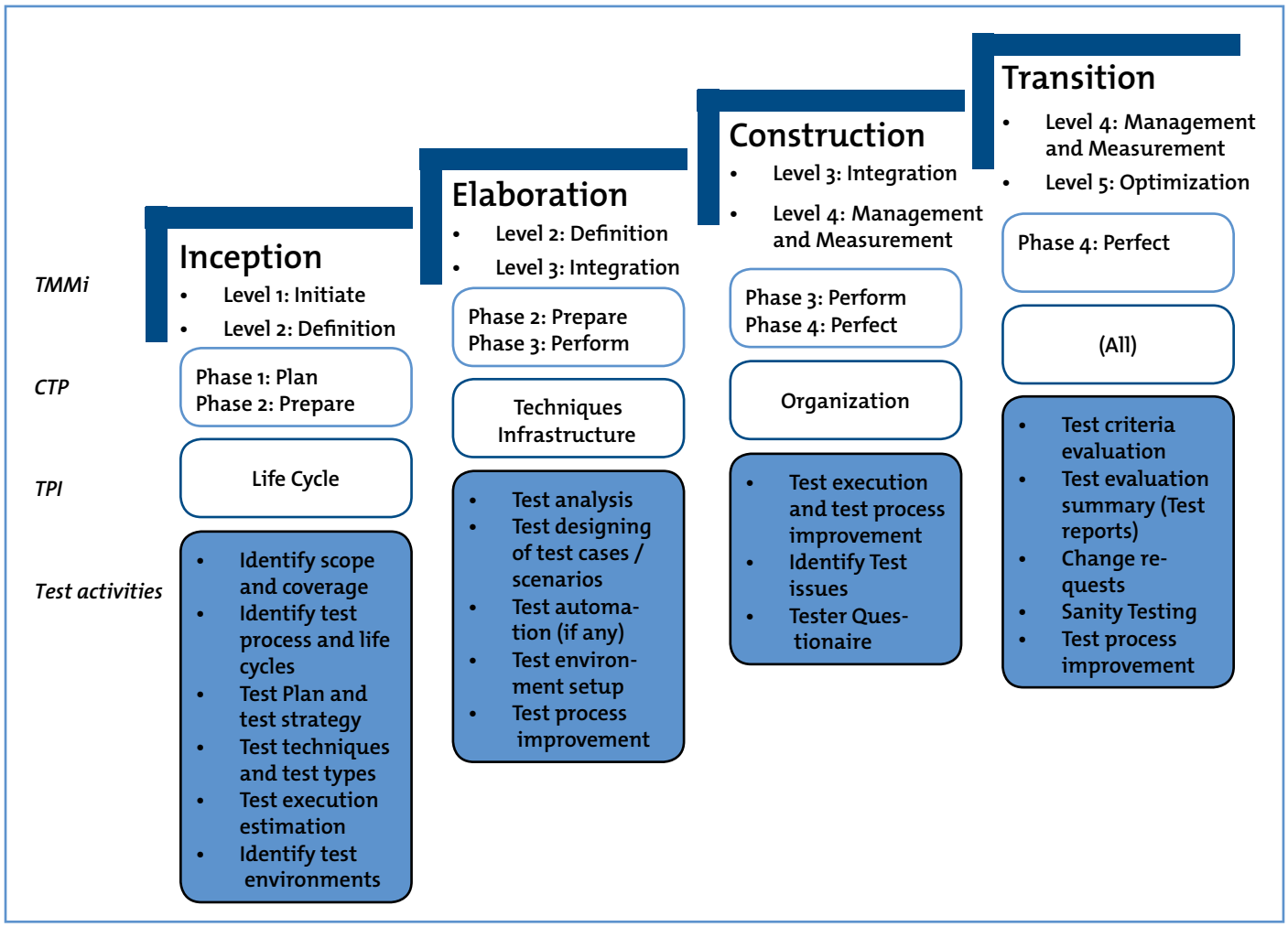


Figure 2: Various Test Process Improvements with RUP

Knowledge Transfer – The Trainer Excellence Guild

Díaz Hilterscheid

From User Stories to Acceptance Tests
by Gojko Adzic

- Oct 10 – 12, 2011 Amsterdam

Foundations of Agile Software Development
by Jennitta Andrea

- Jul 7 – 8, 2011 Berlin
- Jul 11 – 12, 2011 Amsterdam

Rapid Software Testing
by Michael Bolton

- Dec 13 – 15 2011 Oslo

Agile Requirements: Collaborating to Define and Confirm Needs
by Ellen Gottesdiener & Mary Gorman

- first quarter 2012 – tbd Berlin

Testing on Agile Projects: A RoadMap for Success
by Janet Gregory

- Sep 26 – 28, 2011 Brussels

An Agile Approach to Program Management
by Johanna Rothman

- Sep 12 – 13, 2011 Berlin
- Sep 15 – 16, 2011 Amsterdam

Risk-Based Testing
by Hans Schaefer

- Dec 14, 2011 Helsinki

Website:
www.testingexperience.com/knowledge_transfer.php

Outsourced testing @ domain

by Radosław Smilgin

Most outsourcing companies consider their testing services in relation to web and desktop applications. Business in banking, telecommunication, or in embedded software seems to be still an unexplored area.

A few factors must be considered when doing outsourcing testing at domain. What is most significant are people, though we also need to take payment, protecting data and, above all, test environment into consideration.

People

The most difficult question that needs to be asked is whether an experienced tester is able to test all types of software. A natural answer is "no". There are no perfect, versatile testers, qualified in all testing techniques. However, if we consider black-box testing techniques, then what we mostly care about is not what happens within the software, but what is going on at the interfaces. All we need in such a case is a good requirements specification. Although understanding of a client's overall business is always crucial, in the end we test an application which is described as a set of functions and use cases. We do not analyze the application as the most important one in the world, but rather as just another piece of software to be verified. Possessing a good test oracle and a tester to read it should be enough to complete the task.

As a customer you should know the people who are testing your software, as this would facilitate your cooperation with the outsourcing company.

Payment

Sooner or later a customer will ask "how much is it?" The answer is that testing is infinite, so is budget. Over the years every outsourcing company has developed its own concept of how to sell testing which fails to be measured. Nevertheless, there are solutions that may be implemented if domain testing is on the table. You will never agree to get your payment as a fixed price. The cost estimation process will be more difficult at the domain testing, since you have no opportunity to see the whole software inside the production (or production-like) environment. Due to security issues you will not be able to see the whole documentation. Therefore, when asked for pricing, you should obviously stick to time and material which means you will be paid for work you have actually done.

From the customer's perspective, you will not be able to get a perfect cost estimation since you are not providing the full requirements specification for calculation.

Outsourced testing, however, is only a part of the verification and validation processes that you have implemented for a given project. You will not risk success of your final product with a single quality check. You will do this yourself at the development or system level.

Protecting data

Protecting know-how and sensitive data are only one of the major risks related to outsourced testing. Therefore, companies which offer testing services must fulfill and, in some cases, also anticipate some of clients' security expectations. Non-Disclosure Agreements will not be the only tool in protecting the data of a customer. The other are financial penalties which are included in the agreement between the parties.

Some protection tools do not work in some countries. If you care about know-how, you will probably never outsource your project to China, knowing that their consideration for copyrights is at the same level as for human rights.

Test environment

The greatest obstacle for providing testing services for corporations working in specific domains, is the test environment. Are we able to build on our site an environment that is precisely similar to the final production? Probably yes, but with infinite budget only. So how to test an application when there is no test bed? We do not have to. The customer will be the one to deliver the environment or to share expenses for building it with us. With perspective for long-term cooperation, building an own test environment will become part of a business case and, in the end, a customer will be the one who pays for it. With a bit of understanding and goodwill, a customer will provide the environment himself. No one can expect that the outsourcing company will be able to build a simulator for a finance system or telecommunication network. It is much easier to provide secured access to the customer's internal environment. This will also give the customer

an opportunity to observe the outsourcing testers' activity with the application. It will be very similar with automotive or medical devices, or any other embedded software applications. The customer should provide us with emulators of hardware in the first place and then support further testing with first hardware prototypes. We all know the problems with flying on the net pictures of the very first to-be-released models of iPhone. Such situations are quite common when people are low-paid or are not fully controlled as in India or the Philippines. It is also an outsourcing company's job to protect a customer from such incidents. If this is done, the customer's trust will be gained and the company can get long-term contracts.

Summary

The process of deciding whether a company is able to test @ domain, is mostly related to the experience of the people that are the major subset. The other side of the coin is not only the ability to convince that both software and know-how will be secure in our hands, but also to show the ability to provide valuable feedback about quality.

The cost estimation of testing software by the outsourcing company must include building or providing access to the internal environment. However, this will also include a wide risk analysis for all security issues.

> biography

Radosław Smilgin
As an experienced tester with a rich project portfolio and a strong theoretical backup, Radek trains testers and prepares them for certified examinations, as well as supporting many IT projects. Having worked as a Test Manager in some international corporations, he frequently conveyed his knowledge into the fields of practical testing. As an owner of website testerzy.pl, Radek shares his knowledge with other testers that also pursue his idea of spreading testing within the IT branch.

Storing Test Assets

by Suri Chitti

In today's testing world, how to store test assets is anything but trivial or obvious. Two important factors contribute to the right strategy: 1) Avoiding a few common mistakes that are frequently done while storing test assets, and 2) Devising an optimum storage process that addresses specific assets possessed by the team and efficiently makes use of the tools that are at the team's disposal. This article elaborates a little on how to get these two factors right.

Most mature test teams devise strategies that address the general needs of storing test assets. These provide access to the asset, restricting the access to the asset to the required personnel, and ensure that the asset is not lost. They also address the specific storage needs of an asset to the degree required to justify the presence of the asset and to keep it viable. Very few teams succeed in ensuring that their storage approaches do not create issues during operation, and much less succeed in devising efficient processes that work best for them. Ensuring that such issues are not faced in operation due to storage activities largely depends on avoiding a few common mistakes. Designing efficient storage processes depends on the specific needs of the assets possessed by a team and the tools that are at its disposal, and some tips which help achieve this are also discussed in the article.

The first formidable challenge faced by a testing team is to make well known, where to find what asset. Most teams address this challenge by providing newcomers with broad summaries of where to find what. These do little to help because of two reasons. Usually such lists are produced as a one-time activity and are never updated, which means they become more inaccurate as time passes. They are not accessed by non-newcomers and are therefore pushed into oblivion. The best way to go about this is to have an asset map that is maintained (updated as frequently as required) and strategically placed to be visible to all – displayed on the team portal, pasted onto cubicle walls etc. The asset map should tell every team member two important things – where he can find assets that are relevant to his role, and where he must store assets that result from his work.

Storing assets in informal or non–prescribed ways, for any length of time, should be rigorously discouraged. For example, storing assets in shared folders on networks, test computers, an active team member's computer or laptop and other such informal mechanisms should be avoided. Such mechanisms, while restricting the access or use of an asset from a wider and necessary audience, may also encourage silos in test operations.

Where multiple mechanisms are available for storing assets, only one mechanism should be prescribed and enforced. For example, a training manual can be stored in both a version control and a document sharing tool; when both tools are available, it is tempting to utilize both to store the document. Yet this should be avoided and only one mechanism (preferably the document sharing tool in this case) should be prescribed and followed. Creating duplicate sources for an asset is the next worse thing to loosing the asset, because redundancy in storing assets is likely to result in wrong copies (older versions, incomplete versions, etc.).

Test teams should not store assets they do not own. For example, requirements, application builds, etc. are assets required by a test team, but the responsibility of keeping these viable does not lie with the test team. The true need here is to set up processes whereby the right asset is made available to the test team at the right moment. Another folly is to store too many things. Test assets that are no longer meant for use should either be archived or purged. Reports, communications, etc. should not be explicitly stored, particularly if it is easy to pull them out from a test management tool, mail server etc. Logs from runs should not be stored beyond the period they are needed for analysis. Storing too many things has a number of problems. Access to important assets becomes more complicated and unnecessary expenditure is incurred.

A storage process is a framework of rules and practices established by a team to define how and where significant assets should be stored. The table provides the right approach to store a wide range of test assets. These approaches have been gathered from best industrial practices. While they help a long way in establishing sound storage processes, establishing an optimum storage process is not as simple as obtaining the tools recommended in the table, for the assets possessed by a team. An optimum process is not generic. Specifics like the storage needs of the assets possessed by the team, asset growth projections over reasonable periods of time, the tools and licenses available for storing assets, organizational objectives and constraints, etc. decide

Test asset	Recommendation
Project plan for test activities	Should be stored alongside the plans for other project activities like design, coding etc. Content publishing tools should be utilized...
Estimations	Content sharing tools should be utilized.
Test strategy/master test plan/other plans	Content sharing tools should be utilized.
Ways of working documents, templates and forms, administrative forms	These should be downloadable from organizational websites, intra-net or portals.
Tool evaluations	Content sharing tools should be utilized.
Test cases/test data	Test management tools should be utilized.
Test results	Test management tools should be utilized.
Defects	Defect management tools should be utilized.
Test reports	Test management tools should be utilized.
Logs/defect investigation reports, etc.	Logs should be periodically purged from test machines. Defect investigation reports should be attached to defects raised in defect management tools.
Automated test scripts	The automation tool repository or version control system should be utilized.
Automation functional library	The automation tool repository or version control system should be utilized.
Automated run results	The automation tool repository or version control system should be utilized.
Utilities like database files, scripts, etc.	A version control tool, preferably the same that is used to store and build applications to be tested should be utilized...
Training manuals	Content sharing tools should be utilized.
Audit procedures/logs and reports	Content sharing/publishing tools should be utilized.
Test lab access, account request, etc. forms	Content sharing tools should be utilized.

what the optimum for the team actually is. The goal should be to utilize existing tools that belong to the test team, the project team and the organization, and in that order. While space needs and costs may not substantially be optimized by utilizing existing tools, licensing costs of procuring additional tools for the sole purpose of storing test assets should be avoided. Building custom tools or utilizing freeware may not incur licensing costs, but the overheads like effort, training, etc. may not be justifiable either. For example, while a test management tool is the best way of storing test cases, it should not be procured for the sole purpose of storing test cases. Rather, the version control tool utilized by the project team may suffice. In the absence of other strategies, an optimum storage process should strive to achieve proximity to the personnel who are most likely to utilize the assets. For example, automated test scripts can be stored in a test management tool and the automation tool itself may provide version control provisions. Since automated testers work mostly with the automation tool, this should be utilized to store the scripts, even if a test management tool is also available.

> biography

Suri Chitti
is a software test professional. His extensive experience prominently includes testing software in several industrial verticals, automated functional testing and performance testing. He has worked for major companies that are involved in the production of commercial software, across the world. Besides testing, he enjoys swimming and watching the sky at night.

Automation responsibility

by Zbyszek Moćkun

We are in the project meeting and discuss how to speed up the time of software releasing. Suddenly someone from the end of the room shouted: "Let's automate our regression tests". The idea looks very interesting, but during further discussion about automation we found several questions that need to be answered. Before we start to work on automation we definitely have to answer one of these questions: Who should be responsible for automation and why?

The first step is the same as for any project - at the beginning one should define a goal. We need to know what we want to achieve. Do we want to automate only smoke tests (few tests that verify if everything is ok), check all important features (about 20% of coverage), or do we need to automate almost all of the functionality of our application (about 80%). Please note that we can automate even 100%, but the costs of automation for the last 20% will be very high and in most of cases it's not worth to do this (manual tests are cheaper). Below I will concentrate mostly on the most popular and the best approachinvolving 20 to 80 percent of automated tests.

We must not forget about two resources that are always limited: time and budget. If we sum up those (refer to the goal from the paragraph above) we realize something very important – automation is a project. It's unique, and we move within a triangle (scope, budget and time).

Another question is - What kind of project is it? Is it external or internal in accordance with the application (main project) that we want to test? Before I answer this question, let's go back to automation. The task of automating can be divided into two phases: phase 1 - create new test scripts, and phase 2- maintenance (run and keep up to date). The question is - should we treat both of them in the same way?

Of course not - Writing new test scripts can be outsourced to an external project. The author (person who writes automated test scripts) does not need to be involved in the application project since he rewrites regression tests. If manual test scripts are well made, the author does not even need to know anything about the application. He knows the steps and the expected results, which should be enough to created automated test scripts.

On the other of the hand, we have maintenance, which contains activities like: run tests, report results, debug failed tests and update automated scriptsif there were changes in the application. There is no way to do this outside the project, so maintenance can be considered as a type of sub-project (internal project) which should be conducted within the budget and resources of the project. It will be very difficult to make quick changes to scripts (connected with new features/change requests to the application) as an external project. So the changes should be done as soon as they were identified by one of the project team members.

Let's summarize: Writing automated scripts is an external project, and maintenance is internal.

It's time to go back to the second part of the question - who should write the automated tests scripts, and who should be responsible for the maintenance? Should it be done by developers or testers? Does anyone know developers that like testing? I don't think. It's only the one reason but really important (if people do not do what they like, they just change a work or their performance slowly decrease). Another reason against developers is that they would have to learn about the test case management tools, where we keep test cases, learn about the environment and about the tool that we will be using for test automation. Learning new tools is a time consuming process and we have to be sure that our invest returned in long term. Developers should rather concentrate on unit tests, which is their level. So, why should automated tests be written by testers?

Testers are specialized in application testing; they are trained and should have a sixth sense. They know the testing technics and tools, so there is no need for additional training. In addition, when they write the automated tests, they gather knowledge, not only about the automation tool but also about the tested application. Thanks to this they can help us if there will be more work in the project (e.g., complicated features, need of manual regression, a lot of maintenance work in automated tests) or as a backup to testers to minimize the project risk. Learning automation tools is not accomplished in just few days, sometimes it can take weeks. It's crucial not to lose that knowledge. Developers go to other projects, where automation skills may not be desired, but testers' knowledge will be needed again and again for writing

tests but also for maintenance.

If we summarize the above, we see that there are four different ways to write and maintain automated tests:

1. **Developer Internal** - This is a very interesting way, which I tried to introduce in one of my projects. The idea was that each developer delivers newly implemented features together with the automated scripts. Unfortunately, it failed for several reasons. We had to invest a lot of time in automation tool training (the cost was high as we had to train all developers in the team). Another issue that we faced is the decision on who would write the test scenarios that were to be automated. Developers, testers? In the case of developers, we would need to train them in how to write test scenarios and how to use the test case management tool. If we chose testers, we need to find a way for them to deliver scripts to developers (this idea requires them to create test scenarios at the beginning of a sprint, before requirements go into development). Another issue (I think, the most decisiveone for the failure) is whether automated tests are required for all features. There are always other important things to do, like features that should have been finished yesterday, so writing automated tests takes second placeafter a while and automation dies. It is for the same reason that this approach can't be used for maintenance.
2. **Tester Internal** - The idea is that the same team executes functional, regression and other types of tests and spends any free time on automation. From our experience, however, we know that there is no free time because there is always something to do. In this approach, a lot depends on the tester's personality. If he is "lazy", he will automate the most frequently executed tests only. So if we just want to have really poor automated smoke test suites, this approach is for you. For bigger projects it will never finished with success.

   As described above, this approach is best suited for maintenance of automated tests. Testers run test suites, report and debug failed test scenarios. As automated tests should be runas often as possible (continuous integration approach) it is very important to make changes quickly. Testers in an internal team can do this very quickly because there is no need to send anything to an external team, which always takes time. In addition,the external team does not lose time on updating existing tests and can instead create new ones for new features.
3. **Developer External** – The idea is that we create an external team composed of developers with a separate budget, with a defined aim and resources. The use of developers takes over all the minuses mentioned above for the 'developer internal' project. The biggest factor is that developers are not for writing test scripts, but for writing the application.
4. **Tester External** - This idea combines all benefits from internal tester internal and external developer external. Testers do what they are specialized in, it's very easy to manage the budget, and we have a backup for testers. I have managed several automation projects using this approach, and all of them finished with success. Of course, it is always nice to have a developer (it can be rotated role) assigned to an automation team to resolve complicated programming tasks. Everyone knows that 80% of automated scripts can be done by copy & paste using already written code, so there is no need for special development skills (I believe that each tester should have basic development skills).

Why do we need two different approaches for creating and for maintaining automated test scripts?

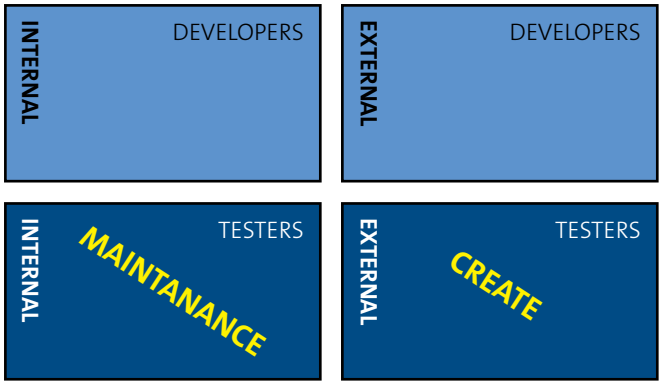


Figure: Automation responsibility square

Maintenance is connected with current application development where new features have an impact on existing scripts that are used for regression tests. As testers (internal) know the test suites and application best they are the ones that can recognize points that have to be changed and can introduce the changes quickly without the need to send them to an external team. It's very irritating to run test suites that we know have failed, because we are waiting for changes to come from an external team. We should note that failed tests which are waiting for update do not test anything, and we should cover this area by manual tests. Please note that it is very hard (and needs a lot of time) to find new errors among several already known. This is why fixing or updating test scripts immediately is very important.

Employing on two teams (an external team which only writes new scripts and an internal team to test the application and maintain automated scripts – it can be even the same person who has divided task in time) results in a process for creating new ones, which is free of interference. We can achieve continuous gains of automated tests and coverage. If we sum this up, it gives us shorter regression time and lower budget for testers in the internal project.

An approach like this gives you clear division of responsibility, ensures continuous growth of coverage, and achieves the aim of a fixed budget without any risk to the application development project.

> biography

Zbyszek Moćkun
has more than 6 years of experience in QA engineering for big as well as for small companies. Currently he works as Head of QA Practice for Cognifide - digital technology agency.
The author has been involved in several projects of different size and methodology where he could work on or introduce automation. His experience was used to optimize the time spent on automation and getting maximum benefits. This is very important, especially in Agile methodologies, which is the area in which he specializes.

Column

Computing in the Clouds: Does it really have a silver lining?

by Ian Moyse

There has been a thunderstorm of growing noise surrounding Cloud Computing in the past 24 months. Vendors, analysts, journalists and membership groups have all rushed to cover the Cloud medium – although everyone seems to have their own opinion and differing definition of Cloud computing. According to the most common definition, it is Internet-based computing where shared resources, software and information are supplied to users on demand, rather like a utility company would supply electricity,water or gas. The term is not new; vendors such as Salesforce.com have provided Cloud services in different guises for many years. Other players have been swift to get on board, including Microsoft, HP, IBM, Amazon and Google, to name but a few. Put simply, users now have the choice of a new way to consume computing power, applications and data. No longer is it necessary to buy software on a floppy disk or a CD. Instead, you can have immediacy of delivery through the Internet for an application you want now. Users have been educated into this way of working with iTunes and app stores, and they've come to expect a seamless link between their locally run application and data and information from the Internet – and at a very digestible and economic price point. Buying a robust, polished application or game for below £1 is now taken for granted. As an average user, you are also likely to be using Cloud computing in the form of webmail, Flickr,YouTube, Facebook and a plethora of other services; storing what you would consider private information in the Cloud without knowing where it is in reality... or even caring.In effect, Cloud has become a very simple and trendy way of describing all things that occur outside the Firewall, whether it be on a corporate network or on your home PC. And Cloud computing is already helping to shape the way we digest IT both at home and in the workplace.

It is simply a new 'form factor', a new way of delivering a solution to a customer. We have seen new form factors disrupting and changing many market sectors already. Think of Blockbuster Video, once the darling of the entertainment world and now struggling to survive against the new delivery factors of Netflix and LOVEFILM. Tower Records, once a worldwide brand, has been put out of business by the ability for users to now purchase music faster and cheaper via iTunes and online music stores. The same trends are occurring in computing.

The extraordinary speed at which Cloud computing has come to dominate the landscape has caught many by surprise. None deny it is the Zeitgeist for 2010 when looking back at the past year. With bold press statements such as "Cloud computing will boost UK economy by £30bn a year" grabbing headlines, it's no wonder it is at the centre of so much discussion and scepticism. With Cloud computing expected to enjoy an adoption rate and growth of between 30 to 40 per cent per year, every year for the next five years, vendors are rushing to launch and push their Cloud offerings. From a customer point of view, it is obviously a good thing to have more choices – although you should be aware that in reality there is a wide range of maturity levels amongst vendors, with some of them taking their first steps and looking for their first few candidates for experimental purposes. The major brands, Microsoft, Google, HP, etc. are all endorsing Cloud as key to their future (Microsoft recently announced the majority of its developers have been moved to its Cloud platform), and in both business and at home it will certainly change the way we do so many things.

Cloud computing offers substantial benefits including efficiencies, innovation acceleration, cost savings and greater computing power. No more 12-18 month upgrade cycles; huge IT burdens such as system or software

updates are delivered automatically with cloud computing and both consumers, small and large organizations can now afford to get access to cutting-edge innovative solutions. Cloud computing also brings green benefits such as reducing carbon footprint and promoting sustainability by utilizing computing power more efficiently.

But be warned... There will be some cloud negatives to go with all the positives. For example, with so much reliance on applications and data stored at the Internet level, what happens when you lose your Internet connection, can't get a mobile 3G connection, or the service itself isn't available for a period? In all cases, due to circumstances outside your control, you cannot access your data, your photos or perform the action when and where you wanted. More worryingly, we have also seen a continued increase in threats coming from the Internet from spam, phishing (fake eBay, bank, etc. e-mails asking you to login in order to steal your details), viruses, spyware and scams. There was more malware on the Internet in the last 18 months than the last 18 years combined. Never in the field of Internet conflict has so much impact been caused by so few to so many. The attackers have gone Cloud and are also utilizing the great virtual computing power base to their advantage.

We are already starting to see malware targeting non-PC devices. It's in its early stages, but it is inevitable that we will see more targeting of smartphones, tablets and perhaps even Apple Macintosh, especially as the data that users send to the Cloud becomes more valuable – and the wider range of applications used to access it grows at a rapid rate. Today's generation of users expect to be able to access their applications from their iPhone, iPad, BlackBerry or Android device, and they expect web-based applications and resources to be readily available to those devices.

We are entering a time when seemingly infinite IT power and information is available to a user on the smallest of devices, on the move and at a price affordable to the average consumer. Looking back, who would have expected that in 2011 you could have a 59p application on your phone which could 'listen' to a music tune being played in a bar and in seconds, using the Internet (accessing massive Cloud power to do the look up and recognition analysis), present to you the details of the track and a 'click here' to pay for immediate download of the song! The chances are that Cloud will accelerate this affordable power still further. Just imagine where we will be in ten years time!

With so much happening so fast, the pressure is on IT experts to not only deliver more innovation, but also greater security to defend it. As devices get more powerful, the Internet faster, the demand and supply of Cloud applications will sky-rocket and the power in the hands of the user will be greater than we have ever delivered before. Remember, with great power comes great responsibility, and your responsibility is to protect the data, your

ID and the information valuable to you. You would not go out and leave your front door open. Therefore, don't leave your 'virtual' front door open when connecting to the Internet and such powerful applications. By all means, be a Cloud user, but be safe and be secure in doing so.

> biography

Ian Moyse
has 25 years experience in the IT sector. He is a published speaker on Cloud computing and sits on the board of Eurocloud UK, the governance board of the Cloud Industry Forum (CIF). Ian has been keynote speaker at many events and has introduced cloud security to a range of leading companies. Ian was awarded global 'AllBusiness Sales All-Star Award for 2010' and the 'European Channel Personality of the Year Award for 2011'.

Testing process evolution

by Antonio Robres Turon

The role of testing within the software development life cycle started in the 1950s, but was very different from today. In the last twenty years the software testing discipline has evolved and progressed inside several software models. In every model, the role of the testing or QA team has changed to work together with the latest software development approaches and adapt to project requirements.

This article describes the testing process and testing activities in several development models showing the evolution of the testing discipline. The models analyzed are:

- Waterfall
- V Model
- Agile methodologies

Waterfall model

The waterfall model was one of the earliest models that included testing. In this model the software development is divided into several phases and the tasks executed sequentially. The model defines several stages that the team must undertake in order to ensure the implementation of user requirements.

The stages in the Waterfall model are:

- **Requirements**: The first phase consists of requirements definition. The end-user's needs and wishes are converted into functionalities for the software to be developed. The team analyzes, determines and prioritizes the requirements for the next stage.
- **Design**: Using the requirements analyzed in the first stage, the project team translates these requirements into software solutions. In this step, the technology and architecture to be implemented is decided.
- **Implementation**: In this step, the specifications are divided into several modules. The programmers implement the source code of the modules using the technology and the architecture defined in the design stage.
- **Verification**: After the source code is finished, the testing team verifies the software. In this phase the team matches the software developed with the requirements defined and looks for bugs inside the code.
- **Maintenance**: In this phase the software is released and delivered to the client.

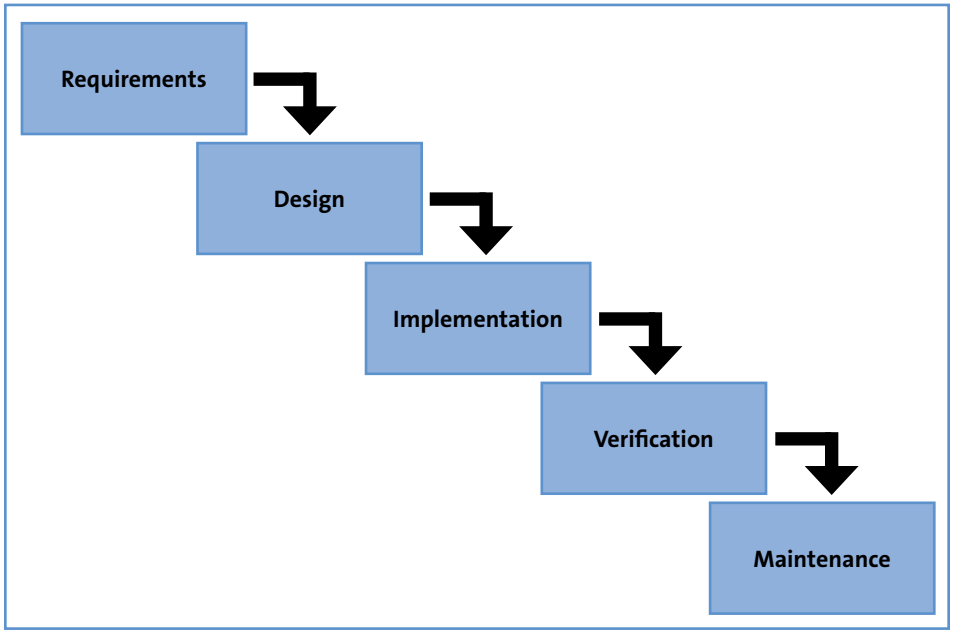


Figure 1: Waterfall model

In this model all testing takes place at the end of the software development process. During the implementation stage, the testing team can plan all the testing activities to be performed in the testing stage.

The principal testing activities in the Waterfall process are test design, test analysis, test specification and finally test execution. The main tests designed and executed are located at the system test level and focus on failures and mismatches between the software released and user requirements.

In the Waterfall process, contact between testing team and development team is not mandatory. Both teams work with the same requirements and specifications and communication between departments is not specifically required. The testing team does not take responsibility for the requirements and design stages; only for the verification stage.

One of the advantages of the Waterfall model for the testing team is the time allocated for every stage, including the verification stage. In this model the test team can execute all the tests with the same software version, and no changes in the source code are made during the execution. Another advantage is the time to prepare the testing stage. During the design and implementation stages the testing team can plan and design the test cases and can prepare the testing environment and the testing data.

The principal problem in this methodology is that errors are detected late in the project. When an error is found by the testing team, the problem can only be fixed by going back to the design of the system. The principal consequence of this is the high cost of defect fixing.

V-model

The V model was developed in the 1980's to address some of the problems found using the Waterfall model.

As mentioned above, the main problem of the Waterfall model is the defect detection in the late phases of software development. The testing team is not involved in the project until the code is finished. The V-model provides a process to perform the testing activities in parallel with software development.

For every development stage there is a testing level to verify and validate the software development activities.

The testing levels in the V-model are:

- **Acceptance test**: The acceptance test is the last stage of software verification and validation and is performed when the software is released to the users. The goals of this level are to ensure that the software released matches with user requirements and to establish confidence in the system. Normally acceptance testing is performed by the user in a "production" environment.
- **System test**: In the system test stage, the testing activities focus on the behavior of the whole software product. This level uses the software specifications and business process to design and execute functional and non-functional tests. In this stage a controlled test environment is required.
- **Integration test**: In this level the interfaces between components and the interactions of the different modules are verified. Integration testing also tests the interaction between the software components and the external applications or hardware. The goal of integration testing is to verify the design and architecture implemented by the programming team.
- **Component test**: The goal of component testing is to find defects in an isolated module (class, objects or functions). Another goal is to verify the functionality separately from the rest of the components. Generally these tests are performed by the development team using "xUnit" frameworks or debugging tools.

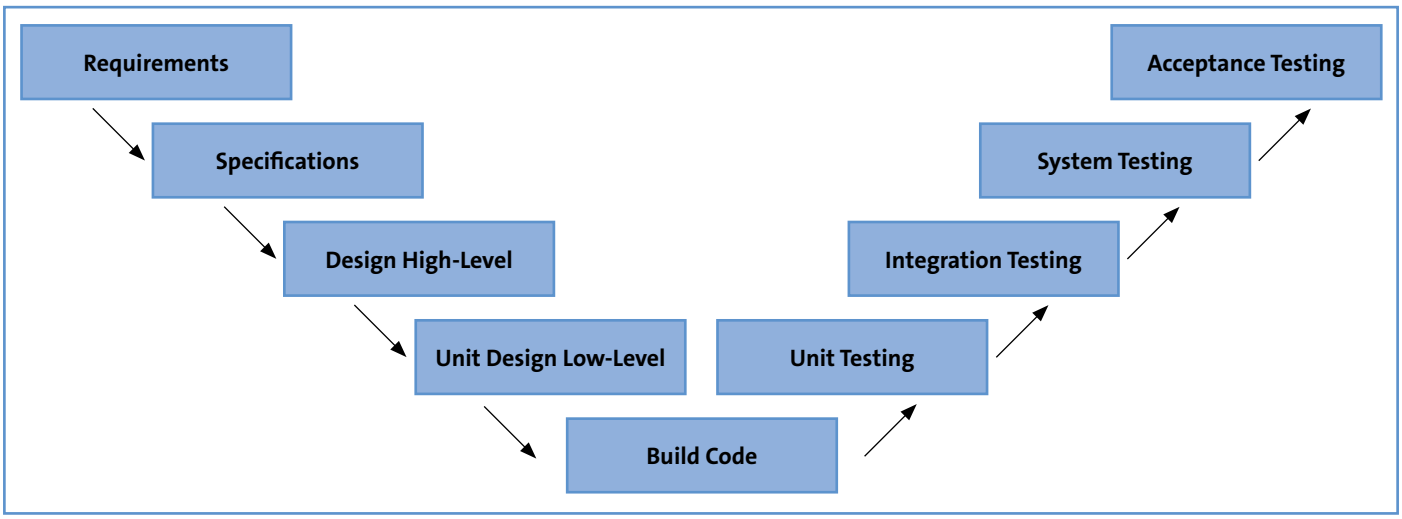


Figure 2: V-model

Other activities in the testing process performed according to the "V" model are the review of the test basis documents. For example, it is good practice to involve the testing team in the user requirements definition, software requirements or system design to find defects or problems in the early phases when they are cheaper and easier to fix.

With this model the testing team is involved in the software development process in the early stages reviewing the documents, and in the later stages performing system testing or acceptance testing.

The principal advantage of the V-model is the possibility of finding errors in all the software development stages and thereby de-

creasing the cost of the defect fixing. Another advantage is the inclusion of the testers in the software development activities, which makes it easier to test design and the testing levels.

One of the disadvantages of the V-model is the amount of documentation generated in the process. If an error is detected in the last stages of software development (for example in the system test), the team must change the source code to fix the error and all relevant documentation, including the design document, specification document and the test documents in all the testing levels. Another drawback of the V-model is the possibility of the re-testing in some components. Some functionalities or components can be verified in two or more different levels, duplicating the work and the documentation generated.

Agile methodologies

One of the methodologies most adopted in the recent years has been "Agile" software development.

Agile software development is based in an iterative and incremental methodology where the requirements are developed and implemented through collaboration among all team members.

The software is developed in several iterations or Sprints which typically last 2-4 weeks and involve daily meetings to review the work of all team members and support the tasks or problems found by other team members.

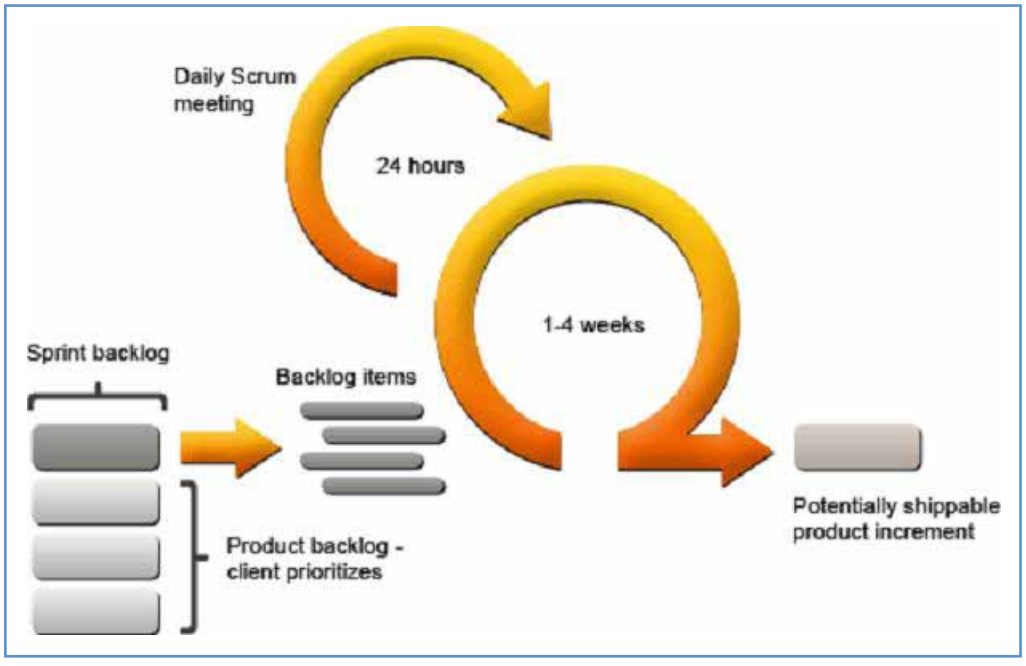


Figure 3: SCRUM methodology

At the beginning of each sprint, the team decides the functionalities to be developed and implemented in the next sprint, including testing tasks. This involves collaboration with all the people involved in the project, including business experts, stakeholders and the programming team. The requirements and specifications are defined by all team members to match with the user requirements.

The test team activities include all activities of the Waterfall-model (test planning, test design and test execution), but the activities are planned and developed in every sprint or iteration.

Manual testing requires a lot of time and effort for the testing team. For this reason, testing must be automated to execute the tests in the next sprints to assure the quality of the functionalities developed in previous iterations. The automated testing can be performed in several levels, such as GUI test, acceptance test, or unit test, involving all the team (not only testers) in the automation activities.

Another important activity during the testing process using "Agile" methodologies is supporting the development process by creating examples or scenarios required to translate the user requirements into specifications for the programming team. To perform this activity, it is important to have collaboration between the stakeholders and the testing team to clarify the requirements, and also collaboration between the testing team and the programmers to help them design and implement the user requirements.

The principal advantage in the "Agile" methodologies is the flexibility to perform changes in the functionalities and the requirements of the software and adapt to user requirements. Another advantage is the possibility to obtain a shippable product in every sprint and show it to the stakeholders to obtain their feedback about it.

Editorial remark: the waterfall and V-Modell have discussed advantages and disadvantages. The discussion on Agile only discusses advantages – there should be some coverage of disadvantages to be consistent.

Conclusions

Testing processes have changed in recent years by using different testing activities to adapt the test teams to software development models.

One of the trends in the software models is the increase of testing activities during the software development life cycle. Adding these activities, the complexity of the testing process is increased. For example, in the Waterfall model the testing team was only included in the verification stage of the project, whereas in the V-model or with Agile methodologies the testing team is involved in all stages of the software development process.

Another trend is the collaboration and integration between the testing team and the rest of the team, including the programmers, stakeholders and business experts. In the Waterfall model the collaboration between testing team and the rest of the team members does not exist. In the Agile methodologies it is one of the most important characteristics.

In conclusion, it can be said that testing processes have evolved in software models, increasing testing activities and collaboration with other departments, such as programmers or stakeholders. This certainly helps in obtaining a final product of the required quality.

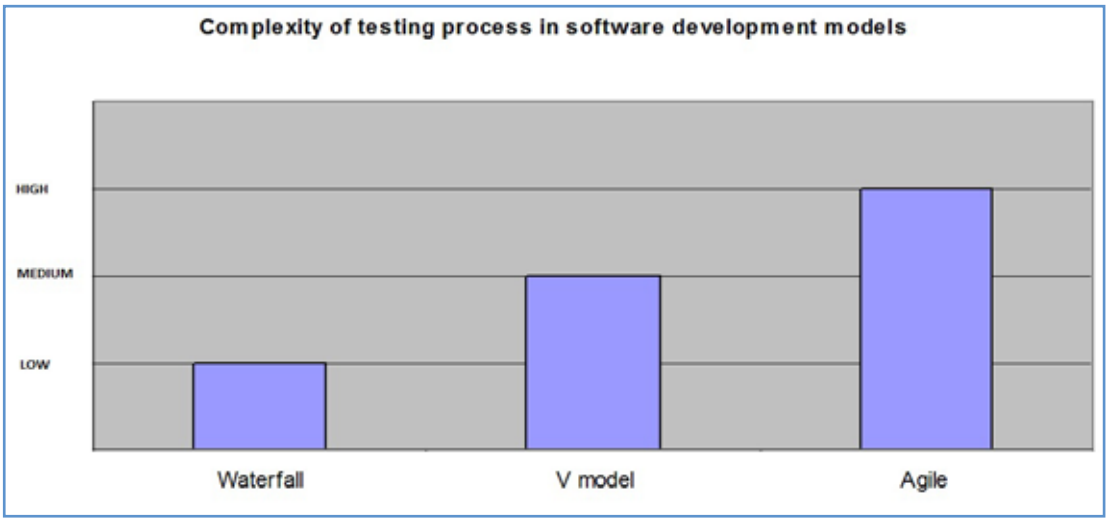


Figure 4: Complexity of testing process

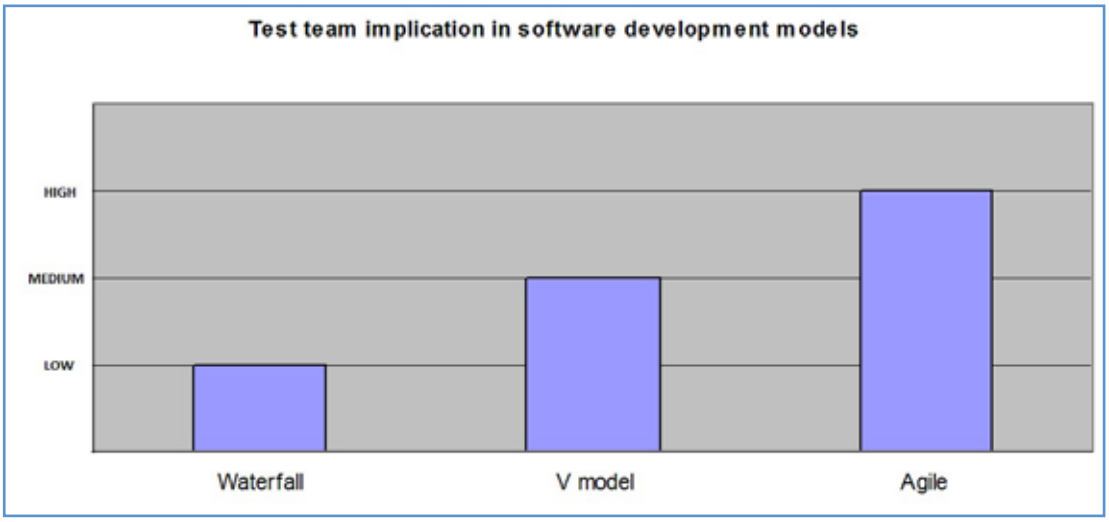


Figure 5: Test team involvement

> biography

Antonio Robres Turon
is a Test Engineer at Diagnostic Grifols in Barcelona, Spain. He studied Telecomunications Science at the Polytechnical University of Catalonia in Spain. He holds a Master Degree in telecommunication administration and is an ISTQB© Certified Tester Foundation Level. He has been working for 5 years in the field of the software testing and quality engineering for companies including Telefonica, Gas Natural or Grifols. His work focuses on the design and execution of several testing projects, mainly for embedded systems and Web applications. He is also involved in the design and development of test automation projects with open source tools. He was a speaker at the last QA&TEST conference presenting in detail about the testing and QA structure of Diagnostic Grifols. He is a writer in the testing blog www.softqatest.com, and also a member of the TestQA association.

How a good test balance can help responding to change

by José Mathias Gusso, Leonardo Oliveira & Guilherme Prestes

When using continuous integration, the organization needs to establish the strategy they will take to test their software through its evolution. This article will cover a couple of experiences that were similar in many aspects, but where the involved teams reacted decidedly different to the changes. Both cases were designed with a structure based on a pre-commit set of tests, followed by an increasing sequence of test levels or phases that were triggered automatically at predefined times. The main variables in this model are **time to run** and **test scope**.

Let's explore the pre-commit phase first. This is when software testing starts (because quality itself begins in the early stages of the project, which is not covered by this article), and the main source of testing at this time is through unit testing. It provides the fastest test response as well as a practical description for future developers to understand the purpose of that code, apart from various other benefits that are not listed here. As small and frequent commits are done, it is also expected that a brief functional validation take place so that we do not inject basic failures into the trunk. This testing is really supposed to be quick, so people don't miss the commit window they have right after updating their workspace.

The importance of a good testing balance in the pre-commit phase is to ensure fast check-ins as well as allowing a stable and updated trunk. One of the most important variables to help defining this balance is the size of the team that will be affected in case the trunk gets broken, depending also on the stage of the project, such as the beginning of an iteration or the days before a major release.

Right after the check-in is when the benefits of continuous integration start to show. It is the moment when all other people have access to your recent changes, and if they were correctly tested and assessed throughout the application, people will enjoy them. Although this seems a magical moment, it is when a simple defect, that had no obvious effect in your local environment, can propagate to the system and cause a major problem in the application, blocking everyone from doing their job until the defects are isolated and removed (either by reverting the code or fixing the problem). Remember that you might not be the only one adding changes to the trunk, so finding the root cause of the failure might require more effort than it looks. So functional testing at this point needs to happen fast and the feedback (i.e. the tests logs) must be very descriptive. This might not solve the problem or highlight its source immediately, but will help everyone in understanding the failure and narrow down the investigation, which results in less time for isolating the root cause.

Choosing what sort of test will be executed here is part of the strategic decision. One of the options I have faced was to build a package and run unit and functional tests right away, while others had a package generation with only unit tests running at this time, leaving functional tests to a third stage. Due to the need for fast feedback, the tests chosen to run here are the most critical for the application, and they are usually a composition of smoke tests plus those that failed recently in later phases.

Talking about later phases, those are the moments where testing gets broader, and more complicated things are evaluated. Usually these higher level phases (typically starting at the third level) allow longer testing, thus providing a good opportunity to fit in non-functional and system testing. If you think about a test that is not really fitting into the two or three first phases because it is not that critical, or because it requires a certain service to be available, or because it is just a simple validation that a given image is shown in its correct position, then the most likely place to fit these tests is in one of these phases. Note that it also depends on the defined strategy and on how the team responds to changes to decide how many of these different phases are available and what types of testing they should contain. Which types of test will be used in each of the phases will not make much difference for the final quality, as long as you keep to the rule that feedback is given before the end of the iteration or before any releases.

One common pitfall with this approach is when organizations decide they will block a release if one single test fails in phases one or two , or if a range of tests fail in the higher phases, (don't get me wrong, this action can be expected as part of a strategy), and if they then suspend the offending test(s), just to allow the release to go through. This typically leads to a situation where these tests remain inactive for more time than they should be, which opens a gap for failures to reach production. The higher the phases, the easier it is for this to happen. So, if you cannot

avoid suspending tests, then you should at least revisit them and resolve the problem by fixing either the tests or the code after the release has gone through.

There are many options to solve problems that come up, and creativity is a good ally when this happens. One can opt for increasing the number of servers or virtual machines per phase to increase the test throughput, or one can also choose to revise and update the priority test list of the earlier phases every now and then. Changing the goal of the later phases can help in organizing when a set of specific non-functional tests will run and in defining the condition in which the issues raised by them will be prioritized in the project. Increasing parallel execution is also a way out of long time for feedback, which helps the team to respond faster to failures.

It is important to note that all the tests mentioned are supposed to be automated. They are mostly scripted tests that need to be executed repeatedly, and would consume a lot of time if they were executed manually. Instead of asking people to do these tasks, it is better to focus on use scenarios and application descriptions to do exploratory testing. The result of a well crafted set of automated tests plus a well executed exploratory test session results in a broader coverage of the system than if testers were to concentrate on manually executing those scripted tests.

The lesson learned was that it is of extreme importance to define clear goals for each phase, and that it is even more important to understand that these goals can be moving targets, depending on the variables involved. Just as important as setting these goals is to learn how to respond to changes and re-align the goals to these changes. For instance, in the pre-commit phase one can be more relaxed if less people will be affected, and/or if the iteration is at its beginning. However, one should be more thorough when reaching the end of the iteration or a release stage, when a large number of teams could be blocked by a simple failure.

> biography

José Mathias Gusso
has been working with software testing for more than 9 years, 8 of them in a large IT company developing and maintaining an embedded web application with administration functions for printing and multi-functional devices. For the last year he has worked as a QA consultant for a consultancy company at a big American retailer validating their e-commerce and internal applications. Most of his experience has been with test automation, and in the last 4 years his focus has shifted to Agile practices.
The author, who is currently working with fully Agile projects, is dedicated to identify and improve the test execution process, principally using the benefits of continuous integration to ensure software quality.

Test Process Improvement, 5 Simple Steps that will Change your World

by Juan Pablo Chellew

Throughout time most software companies have failed to document and streamline their processes, especially their Software Development (SDLC) process. This does not mean that a process is non-existent, on the contrary all companies have a process that they follow, however everyone involved in the process has a different view or perception of what the process is. This lack or differences on interpretations creates gaps and quality issues all around the SDLC. At the same time everyone has a different interpretation of how the process is performing. Depending on who you ask (the department they work for), they will tell you that the process is a failure, that the process sometimes works, that all is great or that there is no process in place. The majority of the time their responses are based on perceptions and personal experiences where no one's perception seems to be the same.

If a process is not documented, then it cannot be repeatable. If you cannot repeat your processes, then you will not be able to measure them. if you cannot measure your processes, then you will not be able to identify what's working and what's not working. And if you can't do this last part, you will not be able to identify problem areas and be in a position to improve your processes.

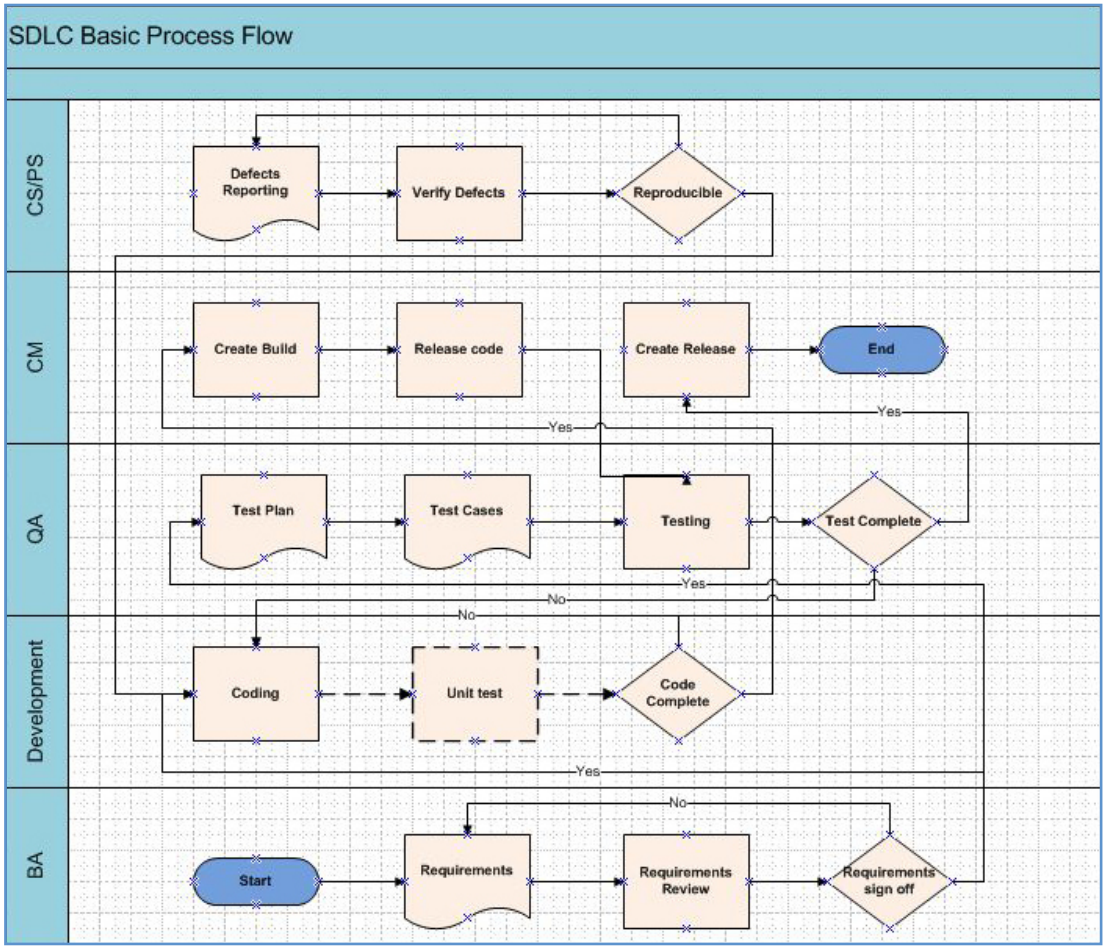


Figure 1 (Actual SDLC Process)

I have always believed that quality starts with a quality well documented process, and for this reason I will share five simple steps that I have learned throughout my working experience that you can follow to establish a world class quality process: first, identify your actual process (as is); second, assess the process; third, improve the process; forth, execute the new process; and fifth, monitor the process.

Let's start with identifying your current process or as-is process. In order to this there are several techniques that you can follow. The one that has best worked for me is using a process map, also known as a cross-functional map template. A process map has two sections, the processes and who is res-ponsible or owns the process (see Figure 1). There are plenty of templates out there, MS Visio has a descent process map template. Once you have selected the template that best fits your needs, you can identify the departments that directly interact with QA. These departments will be most likely Quality Assurance (QA), Development, Software Configuration Management (CM), Documentation, Customer Support (CS), Program Management (PM) and Professional Services (PS). After you have identified the process owners, you will need to conduct a set of short interviews to identify their understanding of their actual view of the overall SDLC process.

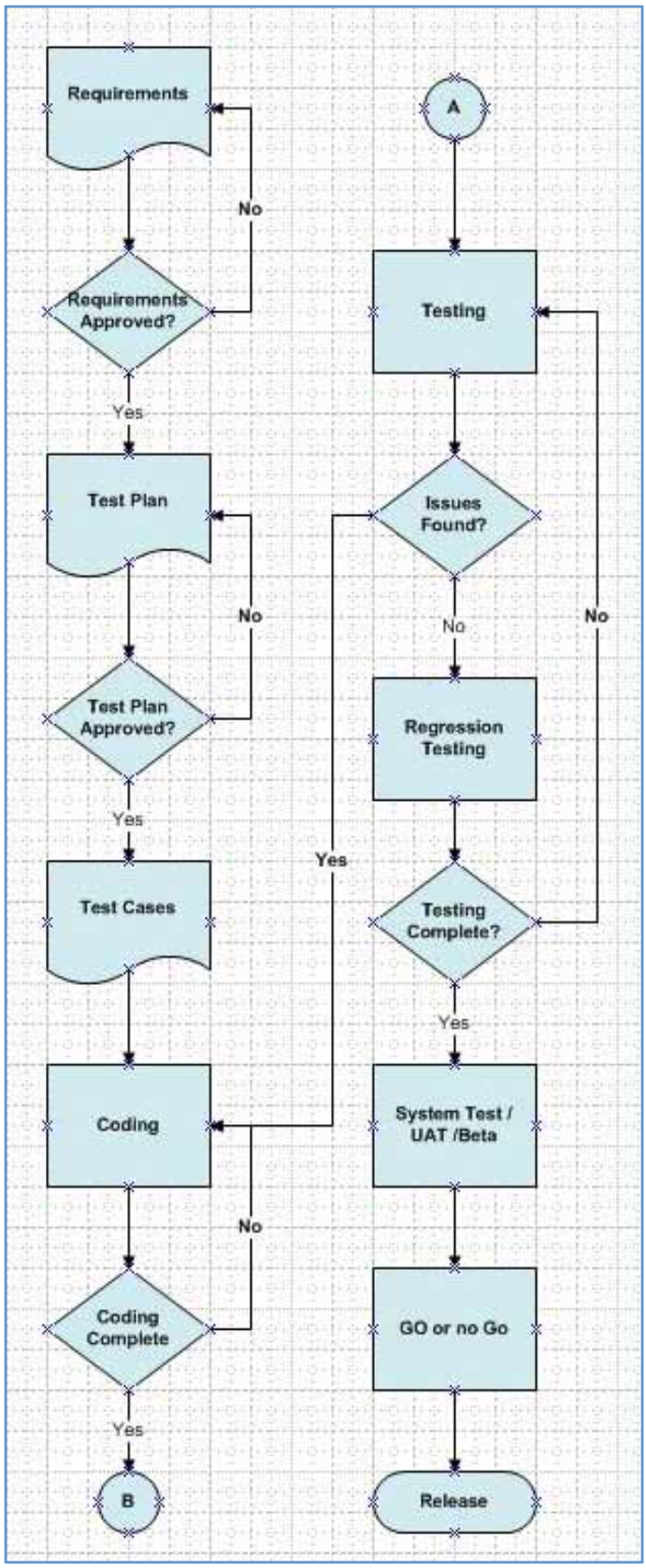


Figure 2 (Basic Testing Cycle)

Once this step is completed, we need to concentrate on the processes for a single department, and since this is a testing article we are going to pick the QA department. For this step, we will interview the QA analysts and get their views on what the actual testing process. For this exercise we will use a process flow diagram, see Figure 2. The goal is to map all the internal processes involved within testing, from the requirements to development to testing to release.

So far we have documented all of the actual processes and departments' interactions that are involved in producing and delivering quality code. What now, you might ask, well with this information there are a couple of steps that we can take. We can dig deeper into the interactions of the different departments that QA interfaces with (top down approach) and identify problematic areas, or we could look at the testing process and see if we can identify areas that might require improvements (bottom up approach). It is up to you to decide what approach you would prefer to follow. I personally, however, prefer to start with the bottom up approach and look into the internal testing processes. This will allow you to have more control and a direct influence on the process improvement areas in case that we might have to make modifications. It could be that after going through the bottom up approach, you feel that the process is as good as it can be, that most of issues lie outside QA. In this case I would suggest that you proceed with the bottom up approach. I have to warn you though that by skipping one of the discovery phases you are relying solely on perception and not on physical data. My recommendation is that you perform both approaches; this way you will in the end have well documented all processes.

For the next exercise, when you will assess your processes, you will need additional information to surface#uncover areas needing improvements, and a second round of interviews will need to be conducted. This time your focus will be on what currently is not working, what needs improvement, where are you spending the majority of the time, where are the roadblocks that prevent you from moving forwards through the process, to give you some ideas. Gather the troops and conduct a simple discovery exercise, e.g. organize a brainstorming meeting and use an affinity diagram using yellow sticky notes, see figure 3. The purpose of the brainstorming and affinity diagram is to identify and prioritize the areas that need improvements throughout your process before you can implement them. Creating groups or subsets of ideas will make the result easier to manage and evaluate. In this exercise there are no bad ideas, and the team should not discard any thoughts brought to the table. The team will come up with ideas as to what is not working, and these will be written down on sticky notes and placed on a board. Next the team will come up with a set of categories and arrange the sticky notes underneath

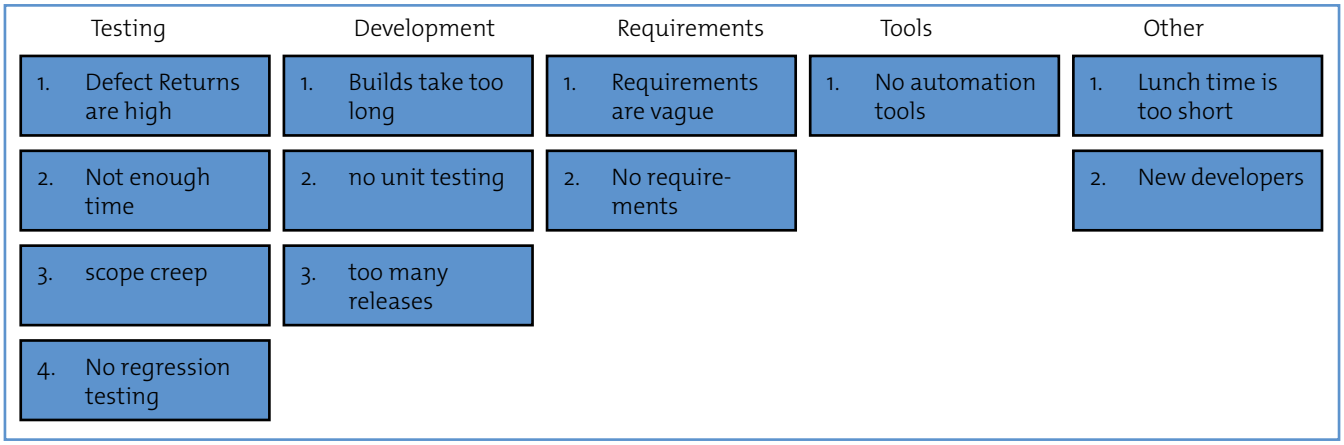


Figure 3 (Affinity Diagram)

each category that they belong to. Once all the issues have been categorized, they will need to be arranged based on their priorities. The priorities can range from 1 to 10, from top to bottom, 1 being the highest priority and 10 the lowest.

Now that we have all this data we need to prioritize what group or category we will start working with first. In order to do this, we need to identify the groups that we have direct control over and those that directly affect your deliverables. This will assure that our changes will get implemented and improvements can be achieved. Once a group has been identified, we will look at our number one item in the list.

For this exercise you select the testing groups and l start with the sticky pad with the highest priority (1 Defect Return Rate is high). At this stage of the improvement process you are going to use a technique call the 5 whys. The 5 whys consist of asking five times why something happens or until you get to the root of the issue. For example: Why do we get so many returns?, Because there are too many defects. Then ask: Why de we have so many defects? Because developers do not do unit testing, and so on until there is no response for a why question. At this point you have reached the root of the issue. Most times it will take no more than 3 whys to get to the end.

With the root cause of one of our issues, we can now go back to our "as-is process" and modify the process by adding the activity that was found using the 5 whys (unit testing) to the process map (see Figure 1).

So far we have identified and documented the actual process, we have assessed the process, identified improvement areas, and before we can execute the process we still need to go through one more exercise to measure the impact of the improvement. In this case we need to measure the effectiveness of adding unit testing in the SDLC, and we need to figure out where best to collect data for analysis. In the past my place to go for collecting data for this type of issue has always been the defect tracking system. Hopefully you have one in place, otherwise you will have to institute one. What should you be looking for in the defect tracking system? Well, you are actually looking for the defects that get sent back from QA to development and compare them to the total number of defects created. This will give you the ratio of returned defects. It is always good practice to capture your data and standardize your methods of data capturing before implementing any changes, so they can be used as a base for the improvements (see Figure 4). As we move forwards, you need to examine this metric in order to get an actual picture of the improvement changes and monitor any abnormalities. If an abnormality is detected, we can use the data captured to dig deeper to identify the root cause of the deviation from the expected norm and take appropriate action to bring the process back under control.

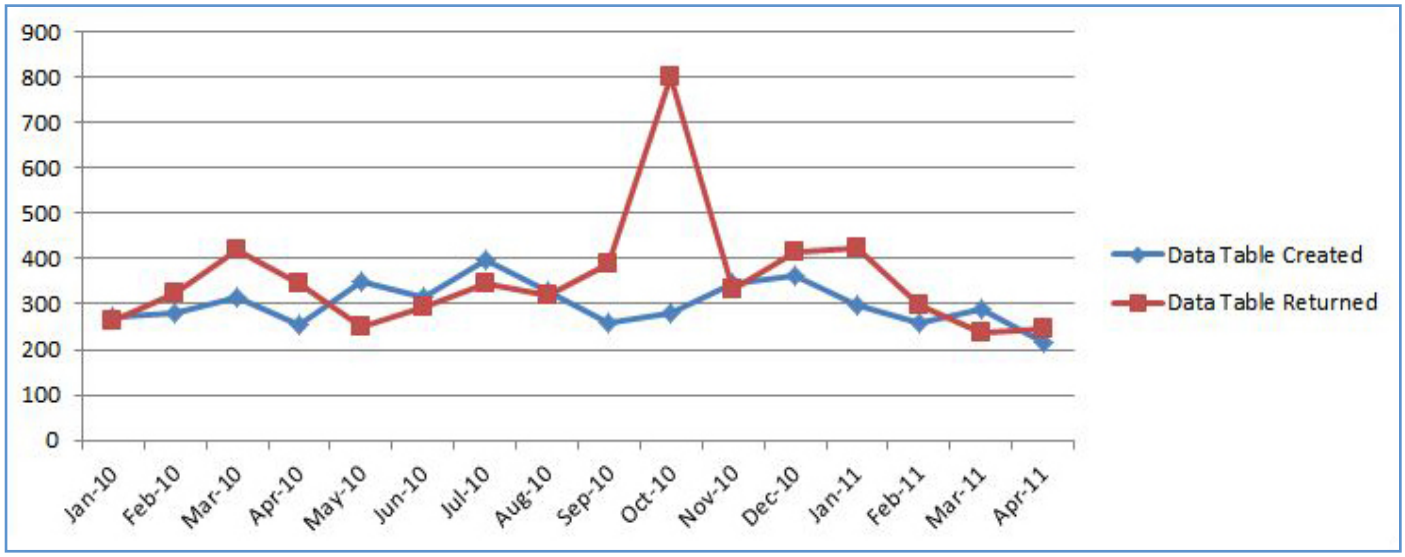


Figure 4 (Control Graph)

The chart above describes a typical rundown or control chart used to monitor the actual process after the improvement (the new process). The new process was implemented in March 2010 and we can see that in March 2010 there was a decline in the number of defect returns and they stayed below the number of defects created until August 2010, when we observe a spike in the graph (abnormality or deviation from the process). After further investigation and doing some data mining it turned out that a group of new developers were added to the group which were not trained in the proper unit testing techniques introduced as part of the improvement process earlier. You can see from the graph that a quick decline in the number of returns happened in October 2010 after the developers were trained in proper unit testing. This spike also identified a flaw in the process, new team members were not being properly trained, and this prompted an additional step to be added to the overall process.

By now you have probably noticed that this process was adopted based on Six Sigma techniques and methodologies. This process was adopted from Six SigmaDAMIC (Define, Analyze, Measure, Implement and Control) and I have moulded and used several basic Six Sigma tools and techniques to make it fit better for the software industry and QA needs for improvements, measurements, implementation and control. These steps should not only be used within QA, but they can be carried out or ported over throughout the SDLC. They are simple enough and yet powerful enough to bring your process under control, providing you with an opportunity for continuous improvements for years to come

> biography

Juan Pablo Chellew

Aristotle once said, "Change in all things is sweet". I believe that controlled and educated changes will produce positive outcomes. Juan Pablo Chellew is currently QA manager for RedPrairie. a company that provides solutions in the area of workforce, inventory and transportation. Pablo has worked in the field for more than 17 years. He has experience leading the efforts in all areas of software testing such as test automation, performance testing, usability, system, regression, Beta, UAT, process improvements through lean development, and implemented software testing methodologies such as Waterfall, A.I.M., RUP, and Agile (Scrum). Pablo's experience involved the following domains: telecommunication, e-commerce, retail, communications, IT and COTS.

Beyond TMMI – Human and non-technical aspects

by Attila Fekete

Nowadays improvement has been renewed and has gained ground again in the form of the re-invented buzzword "innovation". Process improvement is not something what you can briefly describe in a two-page article. It is not possible to even touch all aspects. I have therefore decided to focus on the human and non-technical aspects, which are often overlooked. As most of you know, you can easily get a lot of guidance from TMMi. However, you must keep in mind that TMMi focuses on the "WHATs" and not really on the "HOWs". Let's consider the TMMi maturity levels as a vertical dimension. Each level has some attributes describing what elements your process should have in order to conform to the requirements of that level. Besides this, however, there is a horizontal dimension, too. It is because everything can be implemented in several different ways. And this is also a maturity indicator.

Human factors should never be underestimated. They are more essential than many would think. When dealing with human factors, one should keep in mind that it is more difficult to develop human (inter-personal) skills than many people suspect. Communication with other persons is not standardized like human-machine communication. Everyone is different. One thing that works for one group of people might not work for others. This way there are no ultimate solutions for every kind of issue, but rather alternatives which must be customized case by case. So let's take a look at the human and non-technical aspects, which may have an effect on test process improvement activities.

1. Management support

Proper management support is the base for any kind of process improvement, and it is not different in Software Quality Assurance either. It is necessary to establish a common understanding within the management team on the following topics:

- Importance of human factors: recruitments, proper communication of assessments, etc...
- Relations and dependencies between the maturity of different processes: like CMMI and TMMi
- Expected outcome of improvements
- Costs of the improvements
- Different options for technical implementation
- Time-frame
- Organizational requirements

It is one thing is to establish this management support, but you also need to maintain it by continuously presenting the progress. Visibility of the achievements is essential to maintain this support.

2. Communication

It is pretty important how improvement is communicated both to the management team and to the regular employees. It is important to highlight that it is not about chasing responsible persons for any kind of process defects. Rather it is a common interest to improve ourselves. I saw so many failing or incomplete assessments due to persons at different levels of the organization who were afraid of being blamed or even fired. In other cases, people may just simply be afraid of making proposals because they worry about the acceptance of their ideas by others. I agree that it is not the best situation if people do not dare to share their ideas, but you must handle this kind of issue, too. Even an anonymous idea or comment might be valuable for your organization. Setting up a collaboration tool where people can stay anonymous could solve the problem easily.

3. Awareness

Most people make improvements on a regular basis. They are just not aware of it. They consider only breakthrough ideas to be innovation and improvement. One should raise the awareness that even a short script might also be an improvement or an innovation.

4. Creative environment

It is a general misconception that dealing with innovation and improvements is a task of a small group within your organization. It is best if everyone is involved. Often people with less background might have breakthrough ideas, since they are not prejudiced by the barriers which an expert of the area might be aware of. Sometimes vast amount of knowledge can be a disadvantage, and a lack of knowledge an advantage. A fresh view on any topic could bring up ideas that were never seen before. However, it is not enough to just get people involved; the ideas must be shared

and discussed through some kind of collaboration tool (e.g. virtual idea box). Besides this you mustn't forget that some people are able to invent new things, while others are better at improving ideas. You mustn't underrate the second group in favor of the first one. It would be idealistic if everyone felt the desire to innovate, which however is not usually the case. You should keep in mind that you cannot force people to be innovative. Therefore the management team should create a proper environment for innovation by providing:

- Allocated time
- Tools to support collaboration / sharing ideas / commenting ideas
- Organizational support (coaches who can be contacted if help is needed in elaboration)
- Motivation: at some companies employees are offered a certain percentage of the expected yearly savings achieved by the improvement they proposed.

5. Cultural impacts

There are certain things that an internal assessment cannot easily reveal. This relates to things whose roots are way back in your cultural background, but may extensively affect your effectiveness. Most probably you are not even aware of the existence of those issues. To reveal such issues, it is better to hire a consultant with a different cultural background or - even better - one with multi-cultural experiences.

6. Costs of improvement

It must be emphasized that improvement is a continuous activity that is connected with certain costs. The one thing which could prevent exploiting all potential in improvement activities is the lack of time allocated for this activity. In such a situation people tend to down-prioritize improvement activities in favor of everyday tasks. Considering that for example according to local regulations one is allowed to work 40 hours a week, people should not be counted with 40 effective hours per week when doing the planning. Let's allocate a certain number of hours every week for improvements and innovation. The strategy not to allocate hours, or to allocate hours not on a weekly but on a quarterly basis simply just does not work out.

7. Recruitment

It might sound strange but quality assurance starts with recruiting the right people. Here I do not simply mean people with the right technical background, but people with the right personality and abilities, too. You may have a top -class process deployed. However, it will not take full effect if you hire the wrong people. Unfortunately, the importance of this aspect is often overlooked. Often when trying to find the root cause of issues, this factor is not considered at all. I will not go too much into details since it is a psychological and HR issue, but I have tried to collect the most important skills that a software test engineer must possess:

- Analytical skills and logical thinking
- Eager to learn and follow trends / tools / methodologies (something that is new today might easily be out-of-date tomorrow)
- Curiosity (someone who likes digging deep into the system's inner details)
- Critical thinking and rational enquiry (a good tester is not a believer, but someone with a healthy objectiveness)
- Creativity
- Ability to work under pressure
- Effective communication skills
- Planning skills (especially for Test Managers, Test Leads)
- The ability to envision business situations
- Fundamental QA knowledge

8. Networking

Collaboration with colleagues in your organization is essential. However, it would be foolish not to exploit the potential of collaboration with industry fellows through the Internet. Use the possibilities that the Internet offers. There are several QA-related communities and networking groups, both for discussion, and for sharing ideas and self-training (weekend testing). Make this a habit of everyone in your organization.

> biography

Attila Fekete

After graduating in Information Sciences in 1998, Attila Fekete started to work in the telecommunication industry. Since then he has worked as Troubleshooter, Test Automation Engineer and Test Analyst. He has honed his skills in every area of software testing, starting from functional to non-functional, from manual to automated and from scripted to exploratory testing.

e-mail: fekete.attila@ymail.com

Improving automated regression testing

by Bernd Beersma

With new versions of software rapidly following each other and business demanding shorter time to market and higher quality to market, the way is open for automated regression testing. Why is it, that even despite of all the advantages, test automation is still not a common approach for regression testing?

In this article I will focus on automated regression testing and an approach for successful implementing an automated test tool with a scenario driven framework. This can be implemented for almost all kinds of automated test execution tools (commercial or open source) and all types of interfaces, depending on your choice of tool.

Why automate your test execution?

There are a number of reasons to automate test execution. Here is a short overview of the most common reasons.

Consistent test execution

With automated test execution the human factor is taken out of the equation. Despite of the tester's good effort, it's hard to repeatedly execute tests in exactly the same way, under the exact same conditions. Chance is that not each test step is executed in the way it was executed during an earlier test run. These differences in execution can lead to errors and influence the outcome of a test run. Small errors in the input can have large consequences in a complex test scenario.

Better utilization of time

For testers, improving the utilization of time is probably the best reason for automated test execution. In an ideal world, testers can execute their newly made test script automatically straight away. So they don't have to execute it manually and start with other (test) activities. After the automated test run, they can analyze the results and interpret the outcome, and they can run the test again if needed.

Reduce testing time is also a good motivation for automated test execution. A 'Test Robot' can execute scheduled tests 24/7. This can reduce overall testing time dramatically, but it also can increase your test coverage. Since you can execute more extensive tests in the same time, you will shorten *Time to Market* and/or increase *Quality to Market*.

Better quality of results and completed products

Because the manual execution of regression tests can be experienced as routine and monotonous by testers, it is possible that errors occur in the execution and therefore in the results. This can lead to inferior quality of the produced software. This is eliminated by automated regression testing.

The pitfalls in automated regression testing

Many organizations see in automated testing the solution for reducing test time and test capacity, but when implemented in an incorrect way, the opposite can be the case. Organizations who are not familiar with test automation and had a poor, or no tool selection project at all, often don't realize the impact of implementing test automation in a good way. The most common approaches are to select a tool based on some experience a tester has, a tool review on the web, a good sales pitch or hear saying. One or more 'technical' members of the test team are selected to implement this tool and set up test automation, besides all their other testing related work.

Despite of the many advantages test automation provides, many organizations are not capable of implementing test automation in a structured, effective way. The problems during implementation are often a result of poor knowledge of test automation and the wrong use of this knowledge. Bearing this in mind, test automation leads to a maintenance intensive, automated 'bug' finding solution, that costs a lot and brings little or nothing.

Where did it go wrong? There are a few pitfalls leading to the failure of implementing test automation in a successful way.

High maintenance

One of the most common mistakes is taking the manual test cases and translating them directly into an automated test script using record and playback. The first impression is that a lot is achieved because now all test cases can be executed automatically, hooray!!! However, what if objects in the System Under Test (SUT) change or new technology is used for a new version of the SUT. You need to update all your test automation scripts (record them again). Your test automation project becomes a software development project on its own that may well be stopped and I end up as shelfware. This is the case in a lot of automation pro-

jects I have seen.

This makes this test automation approach an expensive, ineffective and time consuming one, with little or no benefits.

Garbage in, garbage out

A common misunderstanding of test automation is the fact that automated test scripts are of better quality than manual test scripts. Ok, by eliminating the human factor in execution of the test scripts you will reduce the chance of errors through repeated execution of the script during retest or regression test. But still, the automated test script is as good as the tester who created it. For example, if you set up an automated test with poor coverage or one of poor quality, automating this test will only lead to speeding up the execution and will give a poor result in less time. So test automation starts with setting up a proper test case design which leads to good test cases to automate.

Lack of knowledge and expertise

A lot of organizations set up test automation as part of a test project because they want to improve testing. They already have a test policy and a good testing methodology and want to start with the next step, test automation. Often this starts with someone who has heard of test automation as the best solution for improving quality with less effort and at lower costs.

So without any knowledge of test automation, the green light is given and some tester starts with test automation. Tool selection is done by this tester and in most cases the outcome is a tool which is already in use by development, or a tool part of a broader suite of tools which is already part of the organization's software landscape. The tester has to implement this tool alongside other test activities. The biggest mistake to make is to think that you can do test automation in parallel to your other activities. You need to think of it as a project of its own. The tester has to be dedicated to set up test automation. He /she has to have an affinity to tools, and a technical background (former developer) can be a big plus. The tester needs to know the language of testers, but also that of developers. The tester doing test automation is a specialist.

If the tester is not able to focus entirely on the task of setting up test automation, there is a good chance that test automation will fail. What if the tester is only partially available for test automation and spends the other part doing regular testing activities? When a change in the SUT occurs, the tester has to divide the time between two tasks, where the priority is on test automation. If this is not the case due to a lack of time, the quality of testing can degrade, and also testing time can take longer because automated test execution is not possible.

The lack of a structured process and a test automation framework

As mentioned earlier, organizations start with test automation by selecting a tool and then trying to automate as many test scripts as possible. Unfortunately, not much thought is given on how to set up test automation in a way that it is easy, maintainable and scalable. After a while these organizations conclude that hundreds or even thousands of test scripts have been made, without thinking about the use of external test data, re-usable tests or setting up a generic framework for test automation. Missing this is missing out on successful test automation. Maintaining the automation software and creating new scripts will be time consuming. More effort is needed from the test automation team, with higher costs but little result.

So it is important to think about a structured process for test automation (keyword-driven, data-driven or scenario-driven), and to also consider how to set up a good test automation framework.

How to start with a structured and generic approach for test automation

In this part we will focus on creating a structured, generic framework for test automation. In a few steps I will try and explain how you can set up an automation framework that will help you to improve your automated testing. The first important step is:

Selecting the right person for the job

As mentioned in one of the pitfalls listed above, you need to select the right person for the job. Test automation needs a specialist with the right skills and knowledge. He/she needs testing experience, development skills, planning skills etc. You need someone dedicated for building your test automation framework and not someone who is only available part-time.

Selecting the right tool for the job

There are hundreds of test tools for automating test execution. From simple open source web automating tools to very complex, multi-platform and multi-technology tools (tools that use record and playback, tools for scripting etc.). The most important rule to keep in mind is to select the right tool for the job. Start with defining requirements for your tool and start a tool selection project. Don't underestimate this part of test automation; it is a crucial part for success.

Creating a different mindset

We have selected one or more tools for doing the job. The next thing I want you to do is to create a different mindset about interacting with any application. Most of the people I advised on test automation had their eyes opened after I told them the following:

We want to test a chain of applications, from a web front-end to a mainframe back-end. We have a variety of screens to test. In the web front-end we create a new account for an online insurance company. To do so we have a few screens to interact with:

- Press the link *Create new account* on the *Home screen*
- Fill and submit the data in screen *My account information*
- Verify *New account has been created*

Now we need to modify some information in the back-end via terminal emulation. To do so, we again have a few screens to interact with:

- Search *account* on the *Search screen*
- Modify account information and save *account information*
- Verify *account information has been changed*

We just created an account in a web front-end and changed information in the back-end. We identified several screens, also two different technologies, but is there a difference for us? No, because it makes no difference, a screen is a screen independent from technology or environment. This is true for most of the objects that are in a SUT. In almost every application we expect the same, we fill in the fields on a screen and do some kind of action (Enter or F4), and we arrive in a new state. This new state can be a new screen or the existing screen that has been modified.

Now that we have reached this mindset that interacting with an application, any application, is the same for us, no matter what kind of application or technology, we can go to the next step.

Format of a test case

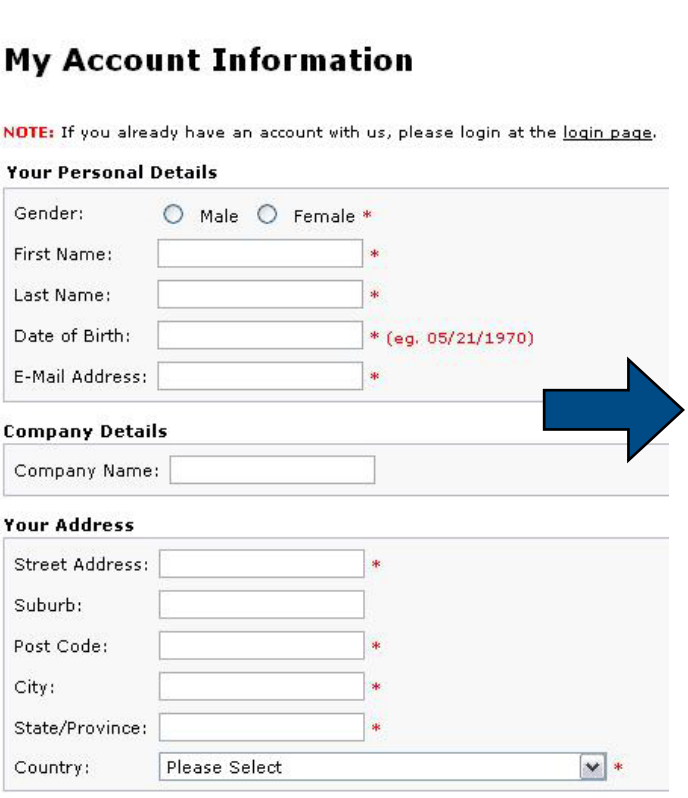
If you want to start setting up test automation in a structured and generic way, you have to focus on two sides, namely test case design on the one side and automation of these test cases on the other. There has to be a uniform way to create new test cases, so they themselves can be a uniform input for the test automation scripts.

One such way is the use of key words or action words for test case design. You create your test step with an action word, e.g. "*Press Submit*", and in the automation software you program all steps necessary to press the submit button in the SUT. The problem here is that you still have a lot of different action words in your test case which can make it complex to automate.

A possible better way is to translate the business process that needs to be tested into separate steps. Let's get back to the account we just created. Several steps were needed to create the account:

- Press link *Create new account*
- Fill and submit the data in screen *My account information*
- Verify *New account has been created*

Once the steps have been identified, we translate them into a logical business flow (in most cases the *Happy Flow*) and create this for example in Microsoft Excel™ or Open Office Calc™. I prefer a spreadsheet program for the overview in rows and columns and for its power to manipulate data like variable dates etc.



Demo	Create an account-Flow 1	
Element	INIT	EDIT
What is New Here		
Log yourself in	~~Check~~	Nee
Create an account		Ja
My Account Information		
Gender		Male
First Name		Test
Last Name		Tester
Date of birth		01/30/1982
E-mail address		#email[squerist.nl]
Company Name		Squerist
Street Address		Citadel 12
Suburb		
Post Code		3900 BD
City		Veenendaal
State/Province		Utrecht
Country	Please Select	Netherlands
Telephone Number		0318551602
Fax Number		0318629996
Newsletter	Nee	Ja
Password		TasQ1
Password confirmation		TasQ1
Continue		Ja
Your account has been Created		
CONTROLE		"A confirmation has been sent to the provided email address"
Continue		Ja

The flow consists of different steps ,and each step represents a screen or part of a screen with input, verifications and actions. For example:

- Fill the Name field with *First Name*
- Verify the default value in the *Country combo box*
- Press *Continue*

Once we set up this vertical flow, you will see that it's a highly readable and understandable way of setting up your test cases. This test case can be read by the tester, the test automation specialist, but also by business users. This is a great advantage, because it can be used as communication with all parties involved in the testing process. Because of the power of the spreadsheet program, new alternative flows can be developed very quickly and a high coverage can be reached.

Now that we have this uniform way of creating test cases and the mindset that every screen is the same regardless of technology or environment, we can start with the automation process itself.

For the automation we use the selected tool for its unique capability to steer objects in the SUT. So this can be a commercial multi-platform tool, or an open source web automation tool, or any other tool. The overall approach stays the same.

Creating generic interaction functions

So, let's start automating test cases. As stated before, a screen is a screen and a button is a button. Keeping this in mind you can start defining your first generic interaction functions to start with a test automation framework. Simple functions that can push a button or fill in an edit field. You can do so by recording or programming (depends on the selected tool) a single steps "*Push Button Next*" and try and make a generic function out of it like "*Push Button(x)*". Where Button(x) is of course any button available in the SUT.

Sie suchen
das Besondere?

Díaz Hilterscheid

If you do this for a variety of objects that are available in the SUT, you build a library with all kinds of these **generic** functions, e.g.

- Push Button(x)
- Fill Edit box(x)
- Select combo box value(x), etc.

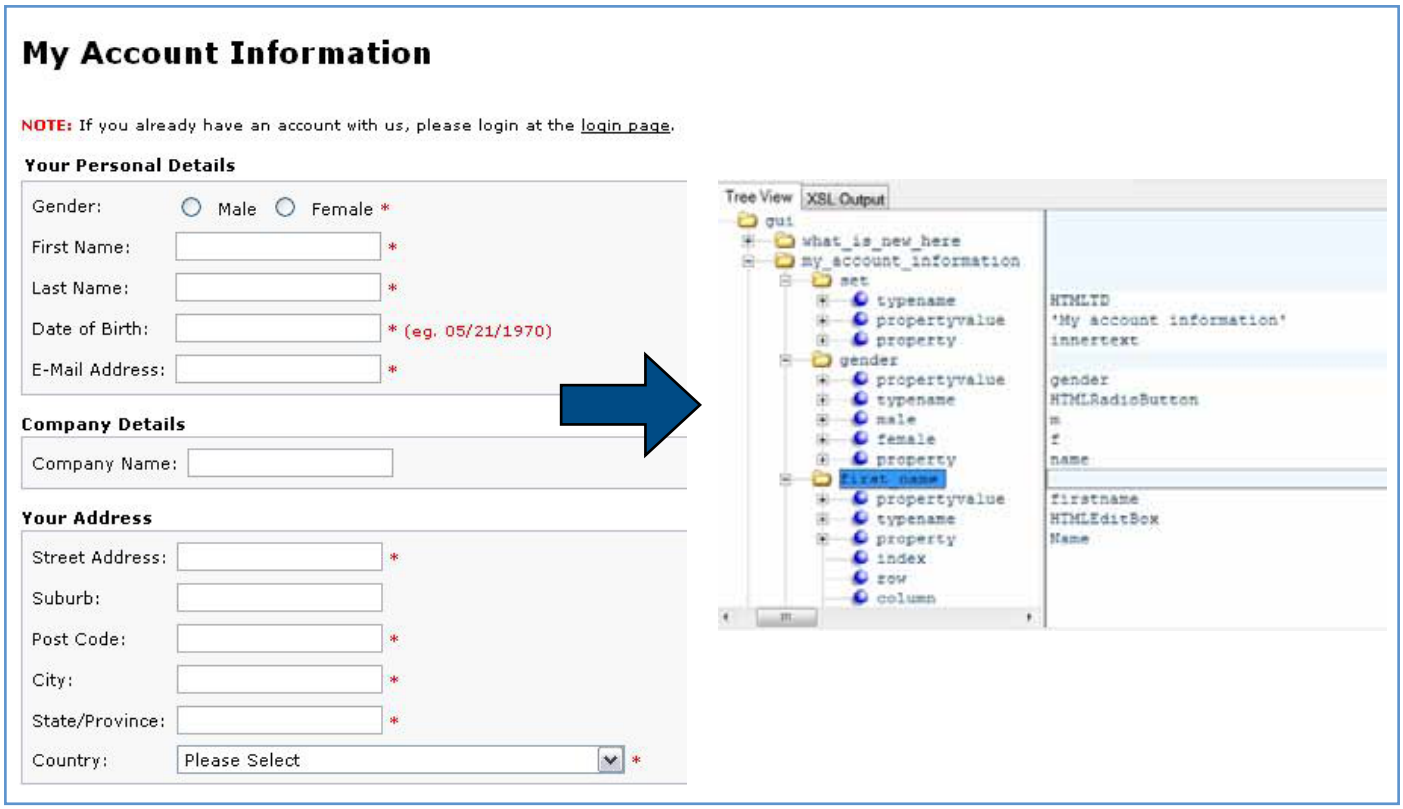
Of course you need to do all kinds of verifications on objects, for example check the default country selected in a combo box when you first access a webpage. For this you create a set of functions like above, but only now for verification purposes.

Now you are able to automate the testing of an entire screen and verify values etc., but where is the next catch?

Interacting with different types of objects

First of all you have to have an idea of how these test tools work. When trying to interact with a SUT, a test tool needs technical information on the object it wants to interact with. This steering information is crucial to the test tool. Many test tools use a repository to store this information in, but I choose to keep it outside of the tool. If we store this information within an XML file (object map), we can maintain it outside the tool when something changes. For example, maintenance can be done by the development team, they know best what technical information is needed to steer an object.

So now we have all the information needed for objects to be steered in this XML file, and the generic interaction functions that need this information are also available. Within the framework you need to build in some functionality to read the technical information from the XML file and use it in the functions that need



this information. The XML functionality will be used for other parts of the framework as well (this will be explained below).

Error handling and reporting

What if unexpected behavior occurs, or objects on your screen have changed? You need to built proper error handling. Building functions for error handling is an important part of your test automation framework.

At this moment let's look back at the primary goal of automated testing: "Automation of test cases". When we manually execute the test cases we just created, we will verify them, and if something is incorrect we log a defect. In the case of automated test execution, however, what if the test executes on a system on another floor or even in another country. How do we know if an error occurs and what defect to create. We need some kind of reporting to these things, so the next step is to build in reporting functionality in your test automation framework.

Preferably, you will want to know what happened, in what step, on which screen and on what object on this screen. I believe the best way is to create reporting functionality which stores the results in an XML-file for each test step executed. Why XML? Because then you can import it to a big variety of tools available and create custom reports.

Final steps

So now almost everything is in place for a generic test automation framework which can help improve your automated regression testing. There are just a few things left to do:

- Get the information stored within your test case in the spreadsheet program into your test automation framework;
- Run the test case within your automation framework.

The next big step is to read your test cases from the spreadsheet program, because all the information you need for the flow you want to test is in the spreadsheet. There are many ways to do so,

Wir auch!

Lassen Sie sich anstecken von der kollegialen Arbeitsatmosphäre in einem starken und motivierten Team. Zum nächstmöglichen Termin stellen wir ein:

Senior Consultants
IT Management & Quality Services (m/w) für SAP-Anwendungen
Deutschland und Europa

Sie haben

- eine fundierte Ausbildung oder ein Studium (z. B. Informatik, BWL, Mathematik) sowie mehrjährige Berufserfahrung als IT-Consultant in einem Fachbereich bzw. in der Organisation eines SAP-Anwenders, eines IT-Dienstleisters oder in einem Beratungsunternehmen in entsprechenden Projekten
- ausgewiesene SAP-Kenntnisse (z. B. SAP ERP, SAP BI oder SAP Solution Manager)
- Erfahrung im Customizing und mit ABAP-Programmierung
- Kenntnisse in der praktischen Anwendung von Methoden und Standards, wie CMMI®, SPICE, ITIL®, TPI®, TMMI®, IEEE, ISO 9126
- Erfahrung in der Führung von großen Teams (als Projektleiter, Teilprojektleiter, Testmanager)
- Vertriebserfahrung und Sie erkennen innovative Vertriebsansätze

Sie verfügen über

- Eigeninitiative und repräsentatives Auftreten
- eine hohe Reisebereitschaft
- gute Englischkenntnisse in Wort und Schrift

Dann sprechen Sie uns an – wir freuen uns auf Sie!

Bitte senden Sie Ihre aussagekräftige Online-Bewerbung an unseren Bereich Personal (hr@diazhilterscheid.de). Ihre Fragen im Vorfeld beantworten wir gerne (+49 (0)30 74 76 28 0).

Díaz & Hilterscheid Unternehmensberatung GmbH
Kurfürstendamm 179, D-10707 Berlin
www.diazhilterscheid.com

Mehr Stellenangebote finden Sie auf unserer Webseite:

- Senior Consultants Financial Services (m/w)
- Senior Consultants IT Management & Quality Services (m/w)
- Senior Consultants IT Management & Quality Services (m/w) für Agile Softwareentwicklung und Softwaretest
- Senior Consultants IT Management & Quality Services (m/w) für unsere Kunden aus der Finanzwirtschaft

Díaz Hilterscheid

for instance create a comma separated file, open a text stream from the test tool and read line by line from the file. However, since we use XML for other parts of the framework, why not export the spreadsheet to an XML file. Because the functionality for reading from an XML file was already created, we can now combine test step information and steering information for screens/objects etc. at runtime.

At this moment a generic test automation framework has been created that is ready to use.

Conclusion

Can we avoid the pitfalls mentioned earlier? First of all the *maintenance* issue. By using a framework as described in this article you can reduce maintenance, mostly because you only need to identify new or modified screens/objects and add or change the technical information in the object map.

The second pitfall, *garbage in is garbage out*. Because of the way test cases are created in the spreadsheet, you always will get the same input for your test automation. This reduces the amount of garbage in, because there is less room for errors or own interpretation of test cases. Ok, spelling errors or faulty test data are still a risk, because the test cases are as good as the tester creating them.

Third, *Lack of knowledge and expertise*. This one is very important, you need to have the right person and the right tool for the job. Skilled test automation specialists are a must, but also make sure you have a good tool selection process. Your framework depends heavily on both.

The last pitfall is of course the one about *the lack of a structured process and a test automation framework*. If you take all of the above into account and you have the knowledge, time and the right tools, you are able to create a structured process and a test automation framework.

You and your test organization are now ready to experience all the advantages that good test automation can provide, and in doing so your regression testing will improve.

> biography

Bernd Beersma
is competence leader test automation and senior test consultant with Squerist. He has over 9 years experience with different forms of test automation and performance testing for different companies.
Bernd holds a bachelor degree in software engineering and became acquainted with software testing during this period.
During his numerous customer assignments Bernd created different frameworks and solutions for test automation with a broad variety of tools. These different approaches led to creating a generic approach for test automation.
As a result of his knowledge about test automation, he also gives advice on how to implement test automation and does workshops and trainings.
Bernd is a member of the TestNet Test Automation workgroup and co-initiator of the Test Automation Day.
His goal is to keep on learning and thus improving his skills on test automation.

Probador Certificado Nivel Básico

Tester profesional de Software

Formación para el Probador Certificado - Nivel Básico
de acuerdo al programa de estudios del ISTQBâ

República Argentina

Docente: Sergio Emanuel Cusmai

Co - Founder and QA Manager en QAustral S.A.
Gte. Gral. de Nimbuzz Argentina S.A.
Docente en diplomatura de Testing de Software de UTN - 2009.
Titular y Creador de la Diplomatura en Testing de Software de la UES XXI - 2007 y 2008.
(Primer diplomatura de testing avalada por el ministerio de Educación de Argentina).
Team Leader en Lastminute.com de Reino Unido en 2004/2006.
Premio a la mejor performance en Lastminute.com 2004.
Foundation Certificate in Software Testing by BCS - ISTQB. London – UK.
Nasper - Harvard Business school. Delhi – India.

Casa Central: + 54 0351 4817951 - **Tel Bs As Argentina:** 011 53685847 - **Tel Londres UK:** 00442033183814
Tel San Francisco. EEUU: 001 14157041515 - **E-mail: info@qaustral.com**

Improving the Test Process

by Jeesmon Jacob

When it comes to quality process improvement, the primary point that comes to the mind of any person associated with the same would be to cut down the time window available. However, is that the only way we can improve the process? By following the well-defined route of quality process and adapting it so that it best suits our requirement,s we can make a considerable contribution to process improvement. The primary focus of anyone wanting to tailor the process should be to ensure that the tailored processes are implemented to the full capacity and that maximum benefit from the same is achieved, which will contribute to high performance. No matter which life cycle is used, the quality process can be tailored without changing its basic structure to complement the lifecycle in use and thus get the maximum benefit.

"We need it at the lowest possible budget" – a common introductory sentence when clients and the top management team are sitting down for a proposed project discussion. When it comes to the actual implementation, however, no one takes the pain to consider this from the whole project perspective with all the cuts, adjustments needed for the testing.

So it always falls to the QA team to ensure that they change their strategy from project to project, time and again, to ensure that the top management's commitments are met. The team should approach each project as a unique one and frame a process that can suit best for the same. 100% commitment from all project stakeholders must be ensured.

Let us consider test process improvement by looking at an actual scenario: Imagine the case of a new project proposal being accepted by the client. Here the following points come into the picture:

1. Top management involvement:

It should be ensured by the top management that the QA lead is informed of the developments and that he/she is informed about all further meetings and communications. By doing so, the lead will get a clear idea about the project from an initial stage, which in fact helps when deciding on many important factors of the testing process. Another commitment that should be extended by the top management to the QA lead is about the availability of funds. This must be discussed at the very beginning so that the whole planning can be done based on the figures available.

2. Resource selection:

Resource selection is an important criterion which will ensure process improvement. Some of the major points that a QA lead should consider whenchoosing the right resources are:

a. ***Domain of the project*** – Here the lead gets an idea of the project and also gets the time to evaluate the resumes of the potential QA team members to assess who has familiarity with the domain and whether he/she is available. Also, a decision on talent acquisition is required, especially if the project is going to be a long running and major one, it may be necessary to acquire domain experts from the industry to carry out testing.

b. ***Risk associated with the project*** – This is another point the lead has to consider when choosing the team. A major distinction can be made between aspects related to human resources and finance.. Depending on the intensity of the risks the lead should allocate experienced hands for performing the testing.

c. ***Timeline*** – A very important aspect of the project is for the QA lead to ensure a sufficient number of resources to handle the project within the stipulated time. Hence, the QA lead must be informed about the deadline of the project and the time available for testing at the initial stage of the project in order to choose the right number of resources.

3. Test criteria selection:

This is possibly the most important decision which has a long-term effect on the testing cycle of the project. Should the project be tracked manually, or should it be fully automated or should a combination of manual and automated testing be implemented? If we consider some of the highly successful projects delivered in the recent past, we can see that a perfect blend of manual and automated testing has helped in making the test process very efficient. Hence at this point, the QA lead should sit down with the team he/she has chosen and discuss the tools and methods of testing to be used throughout the test life cycle of the project.

4. Test plan document:

Now that the base has been set, the team should move on to the next important part, which is the test plan preparation. We have a number of well drafted guidelines such as the "IEEE 829-1998 Standard for Software Test Documentation", which can contribute positively to the whole project by helping us in formulating the test plan correctly, which. Both the development team and the

QA team must participate in this execise to ensure that the deadlines mentioned in the test document can be achieved and any necessary changes must be made with mutual consent. All the details regarding the testing phase must be included in this document. Thus the test plan document should act as a single source to clear all the questions that any stakeholder may ask about the QA process for the project.

Once completed, the test plan must be reviewed and sent for approval to the right stakeholders. After approval, it must be ensured that all activities are carried out in accordance with the test plan.

5. Finalisation of the test matrices to be used:

Along with the creation of the test plan, it is also important to discuss the matrices that should be used throughout the test cycle. Currently, with the advancement of technology, we have a number of matrices available that help in the QA activities. However, we should take a very diligent approach in choosing the right ones. The matrices should cater to the needs of the project. It should not come to a situation where QA resources waste valuable time on the creation of matrices which do not give good results. Hence, the number of matrices that will be used and the templates must be finalized.

6. Conduct testing and update the team regularly:

Start working on the application as soon as the requirements are finalized. Prepare the test cases. This is in fact the time when most of the critical doubts about the functionality of the application are cleared. Every doubt, no matter how small, should be raised to the whole team. The reason is that every question we ask at the initial stage of the project helps us building a clean product. So every QA activity must contribute to the whole project in all its phases. Testing should not be restricted to the actual testing on the build delivered to the QA team. It should be something that circulates around the whole project at all times. QA members should not wait until the first build is released before they get involved in the project. Instead, they should make themselves available at all points, e.g. for client discussions on UI designs, report generation, etc. Every point we note helps the team to make the product better.

7. Update the defect log and discuss with the team regularly:

Here again, once testing begins, the defect log should be passed to the entire team at the end of each testing phase. All possible details to regenerate the issues must be explained so that the developer is in a position to recreate it at his workplace. If possible, meetings must be conducted at least once a week to discuss the issues. And the issues must be prioritized and then fixed depending on their severity and priority to the project.

8. Communicate with stakeholders using metrics:

Regular updates should be given to all stakeholders using the agreed metrics. The cause of any deviation (if any) should be explained to them.

So what's new in here#in this article??

We have heard about these 8 points right from the beginning, and there is nothing special added here. So what am I going to try and prove here? This simple question that is taunting the reader... I fully understand ... let me explain..

Uniforms are used in the schools to identify or group students.. However, are all the uniforms of every school the same? No, absolutely not!!! Here lies my point. The common structure is basically the same, no matter what we do to beautify the same.

So here the process improvement is possible only with the diligent use of the above 8 steps.

And how are we to achieve that??? I am not able to follow a single step when I am working on an Agile project?

Questions again, tons of them... My answer to all this is very straight forward.. Be diligent and use what you need. At the end of a successful round of testing, you will notice that automatically you have followed all of the above steps.

It is not necessary that we use a predefined detailed test plan document. For example, let's define the conditions in a mail and circulate this to the whole team. Let them come up with their feedback and propose necessary changes, and this automatically provides a stage for review. Of course, we can then modify our initial mail content by accommodating all the discussed changes and treat this as the test plan after agreement. So in short, a mail sent out the team becomes your project test plan. Is this not a feasible idea??

So by tuning our process to the needs of the project and by adopting easy and time saving ways, we can accommodate the QA process to contribute effectively to the whole project.

As mentioned throughout the article, we mainatain the basic foundation. All we are trying to achieve is to tailor the basic points and use them in the current life cycle in the best way possible. This tailoring does require a lot of experience in the domain of the project under consideration, the software development life cycle followed, and also the end user requirements. These factors should be marked as primary points when considering process improvements because any process tailoring done to improve the performance will, if it is not understood or accepted by the stakeholders, tend to become a total failure. Hence, in order to avoid these pitfalls, it is very importamt that we understand the project, life cycle and the stakeholders well and then work on improving the process.

> biography

Jeesmon Jacob
is an Electronics and Telecommunications engineer with a MBA in IT & Systems. He is a certified professional in CSTE & ISTQB with over 4 years of experience in the field of Quality Assurance. Currently he is working with the mobility team at UST Global® who are a leading provider of end-to-end IT services and solutions for Global 1000 companies. He has also written a number of white papers on mobile application testing. In his free time, he loves working as a freelance tester for the new builds of Firefox, Thunderbird and Ubuntu.

Improving the Team Approach towards Testing

by Eric Jimmink

Many organizations seek to improve the quality of their products. Books have been written about "Test Process Improvement", usually with the premise of having either an independent test team, or limiting the scope of the improvement process to just the work of testing professionals. Both of those approaches appear to be far from optimal. In a survey[1] amongst teams that were exceptionally successful in producing high quality software, the common denominator was that not a single one used an independent test team.

Team approach

In order to maximize the effectiveness of the test process, an approach is needed which encompasses all disciplines. The driving force, the motivation to improve the test process, can and should come from within the team's desire for excellence. This is entirely possible: specialists from all disciplines want to be able to take pride in their work – both the team's accomplishments, and their individual piece of the puzzle.

There must be some form of unity in the way that all team members approach testing. It will not help if there are substantial differences in individual approaches. Given the short time scales of delivery cycles nowadays, team members must be able to depend on each other – they must be able to pick up where another one left off, and the output of their work must meet the team's standards.

The team members have a common goal, which is to meet the exit criteria for as much of the work as possible. In order to be able to achieve their common goal, the team members must streamline their efforts. For preventing delays, enabling early testing and knowledge sharing are of the utmost importance.

Team members must feel a shared responsibility towards product quality, and act accordingly. Actual participation is a key element here. Showing a genuine interest in each other's work is another. If a team employs practices such as promiscuous pairing, then it is hard not to participate.

Collective test ownership

Let's assume that a team stays together and continues to evolve and improve its working methods. From a notion of shared responsibility towards product quality, and a need to streamline and unify working agreements, the team will gravitate towards a state of Collective Test Ownership.

In 2009, the term "Collective Test Ownership" was coined by Elisabeth Hendrickson. It comprises the practice to place all testware in source control as part of the code base, and a team commitment to maintain all tests as if it were regular, shared code. This means that any team member can alter all tests, coding standards cover testware, and tests may be refactored for future maintainability even if the code appears to be working. A shared responsibility for maintaining and executing the tests means that the testing workload must be shared as well.

Equality within the team

Sharing the workload of testing equally within the team may sound easy. It does, however, require a major change in attitude to embrace collective test ownership – a change which is likely to elicit resistance. Many programmers see testing as the kind of nit-picking work which they deliberately chose to avoid. Yet it is entirely possible that a programmer or an analyst is asked to (enthusiastically) join the effort of executing a test which was not automated! Willingness to do work outside one's own comfort zone is a requirement for each team member, regardless of their role.

What is also important in order to achieve the requisite equality within the team, is that each member is treated as a trusted professional, who does not need to be held back in his or her access rights. Simply give a tester rights to modify all code and unit tests, and allow a builder the ability to delete an entire test script with a single push of a button. Presumably they are all professional enough not to abuse their rights (and everything is in source control anyway). What is avoided by giving full control, is that message traffic is needed outside the code base. The latter only leads to confusion and delays, and is therefore a form of waste.

A team member with insufficient technical skills to implement or modify a test single-handedly, can probably still describe those changes by placing a few comment lines in the code. Make working agreements about the tags to be used in the test code, and engineer the build process to highlight the tags.

Refactoring tests may sound tricky: where is the safety net? Regular code can be refactored in the presence of comprehensive unit tests. Reversing the equation, like some test tools do[2] in what is called mutation testing, the validity of the test suite is secured by the presence of the production code. The best time to refactor a test is when the production code remains unchanged for a while. That is something to coordinate within a team. However, it doesn't have to be a formal code freeze at all. Tests that are due for refactoring could be tracked on a list. To ensure that the code remains constant, check it out from source control along with the test to be altered.

Raising the bar

Once a team has reached a state of Collective Test Ownership, they will be able to monitor their own performance, and eager to pursue continuous improvement. That should come as no surprise. In an Agile context, continual learning and process improvement are part of the game. As a team member, your task is threefold. As an individual you must be eager to learn and improve your game. As a team member you must ensure that the retrospective and other actions for improvement actually take place, and that test process issues are discussed. If a team actively takes steps to improve its results and consequently their own working methods, then that is something to share with management and other stakeholders.

Towards the outside world, a team should be able to demonstrate control of the process, and progress in the work at hand. Both are best shown by providing visibility. If you have streamlined your working agreements, then it makes sense to have some information radiators (references on the walls) to ensure that everyone is reminded what you agreed upon. If you are doing well, what do you have to hide?

The team can do it

Lack of test automation is a problem that will fix itself over time. If all team members are responsible, the right people will take charge of the automation. Give programmers a set of tests that need to be executed time and again, and they will find creative ways to automate them.
Testing challenges should be minimized because testability is taken into account by all team members. This may be very valuable when alternative solutions and architecture decisions are considered.

An integral part of learning as a team, is that enough time is reserved for sharing and documenting knowledge about the work. Fortunately, time spent documenting can be minimized by starting out well: using a ubiquitous language in the specifications (= examples; check-style tests), and using the same terminology throughout the code base.

Time spent on actively sharing knowledge must never be seen as a waste of time. If a team puts the least qualified implementer[3] behind the keyboard, with the expert in the navigator seat, the team is learning rapidly.

It is in the customer's best interest that the team shares knowledge and generates a lot of feedback on each other's work. This allows the team to do a better job during the project phase. More importantly, it ensures a better service during the maintenance stage afterwards.

For teams, there must be a strong focus on maintenance - both during the development project, and after the product has gone live. For organizations, this is actually a significant opportunity: here is where the payoffs are most apparent, in terms of cost savings and customer satisfaction. Cost savings will be there because plenty of tests will have been automated, and knowledge is shared to ensure that multiple people can read and maintain the code. Being truly in control of the test process will have a direct effect on the product quality as it is perceived by the customer. If a team is able to deliver consistent or ever-increasing levels of quality, then doing so builds an outstanding relationship with the customer.

Conclusion

More and more teams are becoming self-managing. If teams succeed in adopting collective test ownership, this implies that they are in control of their testing processes, and that they can take charge of continuous process improvement. In that context, where testing is fully integrated into the development cycle, testing need not be singled out in the pursuit of excellence.

References

[1] http://www.specificationbyexample.com/

[2] http://jester.sourceforge.net/

[3] http://stabell.org/2007/07/13/arlo-beginners-mind/

> biography

Eric Jimmink
is a test consultant at Ordina. He started his career as a software developer / lead engineer. Eric has been involved in Agile projects since 2001. As a person, he is a firm believer in learning by doing, leading by example, and thinking outside of the box.
Eric likes to share his views and experiences, both in publications and presentations. He is co-author of 'Testen2.0 – de praktijk van agile testen', a Dutch book about testing in the context of Agile development. At the AgileOrdina blog, Eric occasionally places fragments of the intended sequel to that book.
Eric's past appearances at conferences include EuroSTAR, Agile2008, and the Agile Testing Days. In June 2011, he will be presenting at Agile Development Practices West.

Continuous Validation of a Data Warehouse

by Adrian Stokes

The journey data takes from source (in our case a heavy legacy system) to the online data store, through the extract, transform and load process, into dimensions and fact tables creating first a warehouse and then an online analytical processing cube to generate reports, is a long one. The multiple gateways and actions performed make attempting to assure the quality of the data a daunting task. However, it becomes a much more achievable task if you break it down into sections and then add a dash of automation.

Test automation has been in wide discussion for a long time. Those that extol the benefits talk of "insurance for software", reduction of manual test time and speed to discover the introduction of breaks in the system. While there is lots of information available, it is predominantly focused on software development of applications, with little consideration to data warehouse development. In some respects we are the poor cousins or next-door neighbors to the family that has just got a speedboat!

To demonstrate that data warehouse development QA can keep up and even surpass other areas, I hope to show that the apparent complexity of warehouse development can add quality through automation and is well worth the effort. Even though data warehouses continually evolve, it is possible to establish robust tests that do not require constantly updating at every new release.

Team Background

The team that was put together to create a data warehouse solution looked to use agile and lean principles and methods to create quality software. Testing as an initial consideration helped the team align development and design as well as focusing effort on what is required to pass those tests.

The team came from a number of backgrounds and includes skills such as business analysts, SQL developers, data modellers, ETL developers and an ISEB qualified test professional. Using the philosophy of iterative development, collaboration and defect prevention activities, the team is predominantly self-managed with project priorities given but task priority managed from within.

The Data Journey

ODS:

Our raw data comes from the legacy system via an RPS (real time processing service) into the ODS (operational data store) on a daily basis. The ODS is updated several times throughout the day with daily snapshots created along the way.

ODS Testing:

Using a series of automated reconciliation tests we confirm value totals, number of rows from before and after the load, and the latest addition is a checksum process as additional inspection of the load.

ETL:

The largest step towards moving the raw data into our warehouse comes in the ETL process. The raw data goes through a series of transformations to allow it to populate dimensions and tables.

ETL Testing:

As this is the highest risk section of the data journey, we concentrate our automated testing here. Each dimension and table is tested independently when created, and those tests are automated to offer full coverage of the process. This is in addition to the data characteristics settings contained within the data warehouse itself.

Each ETL process has at least one automated test associated with it as test coverage here is of paramount importance. Without the confidence of these automated tests we could not be certain of our results further down the line.

Dimensions and Fact Tables:

Dimensions are structured to allow the OLAP cube (online analytical processing cube) to access the data as necessary so data can be 'sliced and diced'. Fact tables allow data to be stored in an easily accessible way using defined data structures built by the data architect.

Dimensions and Fact Tables Testing:

Using both manual and automated tests for unit testing provides more automated test coverage for the regression suite. Fact tables are created to hold specific data and the data characteristics are defined by the table settings.

Warehouse:

Once the dimensions and fact tables are populated, the warehouse is ready to operate. Whether the load consisted of type 1 changes (overwritten) or type 2 changes (new line), the amount of changed and new rows is recorded.

Warehouse Testing:

Again using reconciliation techniques the warehouse load process is verified each time it runs.

OLAP Cube:

An OLAP (Online analytical processing) cube is a data structure that allows fast analysis of data. It can also be defined as the capability of manipulating and analyzing data from multiple perspectives. The arrangement of data into cubes overcomes a limitation of relational databases. Relational databases are not well suited for near instantaneous analysis and display of large amounts of data. Instead, they are better suited for creating records from a series of transactions known as OLTP (On-Line Transaction Processing).

OLAP Cube Testing:

A comprehensive set of automated unit tests confirms the integrity of the cube. These feed into the automated regression suite that is run at each code load and before each release.

Reports:

Reports fall into two categories, existing functionality and new functionality. Reports using existing functionality are considered low risk as test coverage is high. Reports with new functionality are of a higher risk and therefore receive more attention. In either case we work closely with the stakeholders to confirm we produce the right product and take several iterations if necessary.

Report Testing:

Existing functionality is checked via a checklist to confirm spelling, formatting etc. with minimum exploratory testing. Reports with new functionality are tested in depth to confirm the new code works as expected.

DbFit

DbFit is part of the FitNesse application. There is plenty of information on this automated test tool at http://fitnesse.org/ and I don't propose to go into great length here. Suffice to say that other automated test tools, both free and for a premium, are available. To reproduce a quote we gave to Gojko recently that summarized the benefits we gained from using DbFit / FitNesse.

> *"Gojko's DbFit framework and advice inspired us to become a market leader in automated testing in a Business Intelligence environment."*

Cruise Control

Available from http://cruisecontrol.sourceforge.net/ this enhanced our automated build process and as the site advises, *"CruiseControl is both a continuous integration tool and an extensible framework for creating a custom continuous build process. It includes dozens of plug-ins for a variety of source controls, build technologies, and notifications schemes including email and instant messaging. A web interface provides details of the current and previous builds. And the standard CruiseControl distribution is augmented through a rich selection of third party tools."*

Benefits

- At the time of writing we are monitoring the results of over 4,000 automated test artefacts and have reduced our defects by over 50% in the last year (2010).
- Each time new code is checked in, both it and the data warehouse environment it will join are continuously validated with feedback received in less than 15 minutes.
- We are working on improving the format of tests to allow them to be read as living documentation describing and validating how our system works.
- Monitoring broken builds allows us to pinpoint problem areas of our system that can be simplified through refactoring.

Conclusions

It is possible to succeed with automated testing in a data warehouse environment. The team have the added confidence that any impact their work has will be quickly identified and corrected.

While this automated coverage has not replaced manual exploratory testing, it does ensure that the more costly manual work is focused on high risk, high value activities and complements the QA process.

We believe that the effort of creating these automated tests is well worth the upfront effort, especially in the data warehouse realm. They can be run thousands of times at modest cost with almost no physical time constraints. We know that testing takes time. We know that testing costs money. If a little upfront effort reduces both, that has to be a good thing for your company's bottom line.

From a QA perspective nothing is more powerful for demonstrating the end-to-end stability of these processes than picking a single value or figure from a random report. Opening the legacy system it came from, finding the selected transaction or value and confirming it is the same. After undergoing eight significant status changes that is a real confidence booster.

Of course, that is not the end of the story. We can still improve. Coverage, depth, definitions etc. And it does not mean we never find bugs! But bugs are now an opportunity to learn and create an automated test to ensure that scenario is covered and any related ones!

> biography

Adrian (Ady) Stokes
has worked at Homeloan Management Limited for 7 years and is currently assigned to Business Intelligence as their Quality Assurance Analyst. Holding ISEB, BTEC, City and Guilds and ILM qualifications, he is a member of several agile and lean groups.
He previously worked in warehousing, logistics and manufacturing taking on such roles as Quality, Health and Safety and Environment Officer with responsibility for ISO accreditation. A keen cricketer he is his club's Chairman and Vice Chairman of the Upper Airedale Junior Cricket Association and a level 2 qualified cricket coach.

Persistence and humility of a tester

by Robson Agapito Correa (Revised by Renata Tinem)

One day I read a great old book and I saw some words of a master about a parable:

"There was a judge in a city who didn´t respect people. In this city also lived a widow who always arrived during his audience and asked for justice, screaming many times, but was never heard. However, she was persistent and every day she was there again screaming for justice. The judge, however, thought to himself: 'I don´t fear God and I don´t respect anyone man, so why is this widow disturbing me?', and concluded his thought: 'I believe it is better to solve her problem now, so this woman will leave me in peace'."

As testers, we can learn two lessons in this parable: persistence and humility.

We should be persistent and diligent every day, and not give up predefined objectives. It is always necessary to show software failures and mistakes, mainly when we know a lot of people don't believe they exist, and their reproduction is very difficult. Nonetheless, we must save evidence, show there are failures and have a strong argument to persuade our colleagues (developers, managers, team...).

We must also be persistent in relation to testing process improvement and quality process improvement. It isn't easy to implant a new process, but we should take things one step at a time to be recognized by our effort, work and perseverance, showing everybody that the better the quality of the product, the higher will be its value in the market.

About humility (what the judge of the parable doesn't have): we must look at ourselves and realize that we don't know everything, and sometimes we know only enough for our work, but we need to improve day by day. Everybody around us - manager, colleague, developer and tester - has something to contribute, to teach. So, we should always be humble, listen and improve, because today changes in the IT area are big, and if we don't understand that there are many things to be learned and don't study daily, we can lose a lot of opportunities. Someone who knows a lot of things today, may know nothing tomorrow. We need to be humble and have in mind that we don't know everything; that we must learn every day. Sometimes we can learn from those you least expect.

Intelligent is the Plato phrase "I know that I know nothing" in "Socrates Apology". We should always think in this situation, and we must be humble with everybody around us, because the world is turning and who is a teacher now, can become student tomorrow.

To finish, I would like to say that the "great old book" I mentioned at the beginning of this article is The Holy Bible and the master (the master of all masters) is Jesus Christ.

> biography

Robson Agapito
is coordinator of a test team at Mega Sistemas Corporativos in Itu, SP/Brazil. He has worked in software testing since February 2007. At Mega Sistemas, he acquired knowledge and learns a lot every day.
He worked as a developer for eight years, which helped him to see both sides (as tester and as developer). Today, he is able to negotiate with managers and directors, as well as with developers and testers.
He is graduated at Fatec (Technology College in Americana-SP) in data processing, post-graduated at PUC (Pontifica Universidade Católica in Campinas-SP) in System Analysis with an emphasis on client/server, and post-graduated at Fatec (Technology College – in Sorocaba-SP) in Information Technology in Manufacturing.
He worked as a teacher at Fatec (Technology College) in Indaiatuba-SP and he taught Information System Planning.
He teaches at Iterasys, a company specialized in quality assurance and software testing. He is certified by ALATS® (CBTS – Certified Brazilian Test Software), by ISTQB® (CTFL – Certified Tester Foundation Level) and by Scrum Alliance (CSM – Certified Scrum Master).
Twitter: @robsonagapito
Linkedin: http://br.linkedin.com/in/robsonagapito

Improving the Test Process – A Herculean Task!

by Kesavan Narayan

Improving the test process is a daunting task, to say the least especially with what seems to be the exponential increase of tools at our disposal, technologies, and development processes currently available to test professionals. It's quite a challenge to sift through the various bold promises and possibilities that are out there to help improve the test process. The QuickBooks Online® team took a journey to improving the test process as we experienced tremendous growth in the past year in our business. Our team had two key attributes that I believe aligned us for success as we embarked on our journey of improving the test process: an encouraging manager along with a very engaged and collaborative team. Perhaps sharing our journey may help expose some opportunities for your own team or at least create some dialog for your team as they consider the Herculean task of "improving the test process!"

Improving the Test Process with Automation

Test Automation often seems to be the elixir of all ails when it comes to curing the ailments of a team looking for test process improvements. In many ways it is an amazing lever that can be used for such desired test process improvements. However, as experienced tests professionals can attest to, automation efforts without careful consideration and carefully thought and sound architecture can actually be a deterrent to improving the test process. We've all seen many automation initiatives that start off with all the promises in the world but end up costing more than the benefit gained. This phenomenon typically occurs when automation is difficult to keep up with in terms of triaging or maintenance. In addition, sometimes the automated tests tend to be seldom the "complex" test sets and focus only on the more simplistic test sets.

It's clear that test automation when given careful consideration in its design and architecture can be the key to improving the test process that results in the following clear benefits:

- It allows test engineers to focus on more complex testing of the non-functional quality attributes (performance, scalability, security, usability)
- It creates the ability to execute tests more often and in a continuous manner build after build.

Given that statement, the more important issue of improving the test process through automation becomes the ability to ensure you have test improvements through automation as opposed to glorified automated test "suites" that run and act as mere data points and lay disconnected with the customer facing product.

Let's take a few key characteristics of using test automation to improve the test process on the QuickBooks Online® team. The Automation team looked to two different frameworks to address automation coverage for 2 different areas of the SAAS product we offer. The "Fitnesse" Framework was leveraged for the Web Services layer and WebDriver on top of an internally built framework was utilized for the User Interface test automation. In less than one year, we've seen the maturation of each to the point of both efforts truly improving the test process for our team. If we look at these two successful independent automation initiatives, the common characteristics were the following:

- Implement effective triaging techniques (quick root cause to failure for a failed automation test case): The Automation team dramatically reduced the time to triage as one of the initial attributes of the Fitnesse test framework for the Web Services test automation. With the web services layer having a very "high" volume of automated test cases, this was a critical design element to allow for scale.
- Resiliency of the test cases to limit false positives or false negatives: particularly in the UI automation this became a challenge that the Automation team handled very well in the framework to allow for confidence in the test results through multiple executions.
- Emphasis on the documentation of the test cases to automate the complexities of actual customer workflows: Clear documentation of the complex end customer workflows greatly aided the Automation team to access the viability of the respective frameworks early and augment these to provide solid automated test cases that actually maintained tight domain context.
- Abstraction layers built into the frameworks to allow for flexibility in tool selection given a growing support matrix (consider the various browsers and mobile devices as examples of our support matrix): In the case of the UI automation framework, the framework handles a lot of the timing issues

inherent in UI automation as well as other problematic issues to solve for test developer productivity.

When looking to "Test Automation" as a path to improving the software process, definitely look to some of the above attributes as critical components during the Test Automation design phase. With careful consideration, teams can very well rely on "Test Automation" as the elixir of all ails!

Taking a "Test Automation Day"

A practice that has also worked very well and can be used to improve the test process is the analogous of taking a vacation day – applying that phrase we take a "Test Automation Day" – the internal term that is utilized is "Test Code Jam", a phrase to characterize the concept of giving up a day or dedicated time per week uninterrupted to focus on developing automated tests suites as a team. This brings the automation team and the functional test team into one room, and the end result is to develop as many automated tests as possible without interruption. It may seem like a trivial exercise, but what we've found is that great synergy arises between the automation framework developers and the functional test engineers. There's also not far to go for the consumers of the test automation framework or the test developers when looking for assistance in building up the automated test cases. The benefit for the test framework contributors is also felt as they get more in-depth product knowledge to help enhance the test framework, or also introduce more business context into automated tests. In addition, there's no discrimination as developers also are invited and can not only get a feel for the test automation framework, but also get a grasp of some testability challenges that they may be able to solve for the team with tweaks to the production code that's being tested. Taking a "Test Automation Day" is a simple concept but, as you can see, it improves the overall test process by increasing the test automation coverage as well as bringing about the critical team collaboration required to improve testing downstream with possible testability insights.-.

Improving the Test Process with Collaboration and Applying Best Practices

QuickBooks Online® is a SAAS business along with being a "global" company, so we experience a lot of the challenges that many test professionals feel day to day. There is a fast pace to the release cycle along with challenges in working with our global counterparts. Consider the benefit of improving the test process by providing the team with a healthy way to manage the challenges along with encouraging the concept of continuous improvements and utilizing "best practices."

Conduct a"Release Retrospective"

When considering continuous improvements, we tend to overlook the "Release Retrospective" or what some call "Release Post-Mortem". We've found these activities for the team to be very productive, since we typically cannot improve the test process without understanding the opportunities that may exist for improving the test process. This activity is critical and speaks to the intangibles of each team and their respective team dynamics. If the release retrospective is entered into by the management and the test team with an open mind to conducting a critical review with a strict focus on eliminating blame, the result is typically a very open discussion with team members that leave the meeting empowered with tangible items that could improve the process for their next release. Many teams shrug away from a formal release retrospective due to the fact that test teams often work with remote counterparts across the globe, and with time a valued commodity for all test teams, a release retrospective is considered a "Time Sync". The way that the QuickBooks Online® team combats this is to "have the discussion" in any mode possible, and to ensure that the discussion or topic is reviewed. We've discussed this at team meetings that are usually scheduled but allocate a portion of time to do a quick retrospective, or have even considered doing it via email where the thread ends up opening up many thoughts and ideas. The point is to make the activity itself a priority using any communication vehicle possible and not to have a bias against it due to time or logistics, since the outcome is invariably always invaluable to "improving the test process."

Utilize "Risk Based Testing"

As complexity grows and test teams find the resulting customer support matrix for validation large in scale, a simple exercise of risk based testing is also an excellent method to improve the test process. Experienced test professional can attest to the fact that having an exhaustive test philosophy typically will not scale. Given the nature of the testing landscape being so dynamic and ever growing in the numbers of combination of browsers to support or even mobile devices to test, more thought has to be put into the concept of risk based testing. To partake in "Risk Based Testing" the team needs to be collaborative and engage with the development counterparts along with their cross-functional counter parts in operations as well as product management. The QuickBooks Online® team looks to data as another key input to the risk based approach when considering things such as the browser type distribution amongst our users. Revisiting the basics of "Risk Based" testing and applying it against your specific product is an excellent way to improve the test process by reducing the test cycle and optimize by executing only the most critical tests sets. What we've found is that the test team is more engaged and confident, considering the weight of the "support matrix" is distributed amongst the team and not a sole decision made by testers.

Stress Early Defect Detection

A best practice we embrace is "Early Defect Detection". Consider this practice in the domain of collaboration since it requires a lot of team effort to implement this fundamental best practice which all software engineering professionals strive for. Test professionals are intimately familiar with the cost of finding a defect late in the SDLC (software development life cycle). Instead of speaking about the business value of the savings of finding issues early in the cycle, let's focus on the various attributes that help in early defect detection and result in improving the test process. Unit testing is stressed on our team to aid the test team by allowing the test team to focus their efforts on more complex use cases as opposed to mundane functional tests. The collaborative component that is key in this respect is a close cooperation with the development team or counterpart early in the development cycle and to test often against early iterations of the release as the feature matures. Without digressing into Agile best practices, consider early defect detection as a fundamental principle in improving the test process. Look to areas where the test team may be idle and brainstorm as to how that time can be more effectively used by either leveraging prototypes, UI mockups or working off a developer local build to get that valuable early test feedback. Combat formality with open collaboration and strive for early defect

detection as it will surely improve your test process by allowing for more predictable releases and less heroic measures by the engineering team by limiting critical issues that appear late prior to the end of the release or, even worst, post-release in production.

Improve the test process for your team

Irrespective of the path you may take to "improve your test process" for your respective team, consider the value in looking at the test automation strategy, team dynamics, product domain as well as the most important component - your team members - when attempting to advocate change. You'll typically find that test process improvement initiatives are difficult to sustain if the engineering team does not feel the benefit in some tangible fashion. Be bold and push forward the test process improvements to your team, but always find ways to measure or quantify the resulting value of the given efforts.

> biography

Kesavan Narayan *has over 15 years experience in software quality assurance and has worked on various products spanning a variety of domains from Computer Aided Design (CAD/CAM) software, web performance management, and more recently small business financial accounting software. Kesavan has a passion for research into Agile software development methodologies that consider test developer productivity and unique ways to produce high quality software through short iterative development. Currently, Kesavan is working at Intuit, Inc. on the QuickBooks Online® team in Mountain View, CA.*

Things a Tester can Learn from a Programmer

by Ralph van Roosmalen

There is often a healthy rivalry between testers and programmers. Programmers never read the specifications, never test their own software, it always works on their machines. Testers always nag about details, act as if they are the customer, and bother programmers the whole day with their questions.

I used to be a programmer and I'm currently responsible for testing software in a development organization. No, this is not degradation. I discovered that it has advantages as a tester when you used to work as a programmer. I would like to share those advantages with you. I hope they will inspire you to think about improvements and try to implement new improvements in your organization.

Code Reviews

Most programmers do code reviews on their code. It is a very good practice, and I think almost required when you do not do pair programming. The goals of code reviews is to improve the quality of the code and share the knowledge of the code.

As tester you should also implement reviews, review your test cases with programmers, customers and requirement engineers. Additionally, also automated test scripts could be reviewed.

When you organize code reviews, don't focus only on what the code does. Make sure every reviewer has a different role, for example, one person could be the scribe, one person looks if the code complies with the guide lines, and one person looks if the code is understandable. This is a normal approach used by formal inspections.

Programming

Test automation == programming!

If there is one thing that programmers (or at least most programmers) are good at, it is programming and organizing a software project. Depending on your background as tester, you have to admit that most programmers have more knowledge about programming than you.

Use the knowledge of your programmers when you start with test automation. Ask one or more programmers to become part of the team/project group who will start with the test automation. This has two advantages: first they become also committed to the project ,and secondly you are able to use their knowledge about software projects.

When you set up test automation you should for example think about:

- Version control, i.e. where do you store your test scripts?
- Applying labels, i.e. do you label your scripts when you release software?
- Project structure, i.e. do you create some kind of framework?
- Project structure, i.e. are you able to work with multiple testers in the same project?
- Language, and how do you handle other language? Do your scripts support multi# languages?
- When do you update your test automation tool, at the start of a new release, during a release, one day before the end of the project?

Brett Pettichord wrote several excellent papers on test automation.

Good enough could be good enough

This section contains (some) prejudices, which I apologize for beforehand. However, sometimes life is simple when you use prejudices.

Programmers don't care too much about the details. They focus on the main requirements and don't spend too much time on all the exceptions. As a tester it is your job to think about all the exceptions.

Sometimes (I know I'm walking on thin ice) testers focus too much on the details and exceptions. Most software should be good enough, software doesn't have to be perfect!

For Gerald Weinberg quality is: „value of a product", which was extended by James Bach and/or Michael Bolton, who see quality as: „value of a product to someone who matters". With this description, it is possible to deliver a product that contains bugs in

exceptional cases/some areas. If users never use those areas, they are not interested whether you as a tester spent too much time in testing those areas.

Be careful as a tester not to spend too much time on testing everything. Act as a programmer sometimes and don't spend too much time on testing in all cases.

(Ok, start flaming me on Twitter (@raroos) about all the assumptions I made....)

Pair programming

One of the good practices of Extreme Programming is Pair Programming. Pair Programming is code created by two people working together at a single computer. Pair Programming has many advantages, and it's not inefficient.

As tester you could work in pairs in two areas, pair testing and pair programming.

Pair testing is testing of software by two team members sitting behind one machine. One team member is in control of the mouse and keyboard, the other one makes notes, discusses the test scenarios and asks questions. One of the two should be a tester and the other one could be a developer, business analyst or another tester. Pair Testing is not intended to execute formal test cases. Use exploratory testing when you do pair testing.

Pair programming can be applied when creating test automation scripts. As mentioned before, test automation is just like programming. If you apply pair programming when writing automated test scripts, you directly review the code, exchange knowledge and improve the quality of the test scripts.

Test Dojo's

A Coding Dojo is a place where programmers come to improve their skills. As important as improving skills may be, it should also be fun! The idea is that people meet for a time boxed period, with a computer attached to a beamer or desktop sharing tool installed and a programming challenge. During the session the group works on the challenge. It is time boxed, so when the time is up, the Dojo ends.

In a Dojo everyone is equal, there are no differences because of experience. A junior person could learn from an experienced person and an experienced person could learn from the fresh look of a junior person. Everyone should be respected for their knowledge and contribution to the Dojo.

Programmers improve themselves by organizing a Coding Dojo, testers could organize a Testing Dojo. To increase your knowledge as a tester, you should meet with other testers. When you share your ideas of testing with other testers, you will receive feedback. Feedback that makes you think about your testing approach. You actively work on a challenging testing problem, meet with other testers and take your own responsibility for becoming a good tester.

Be lazy

A good habit of a good programmer is laziness. He hates repetitive tasks and always tries to automate boring and/or repetitive tasks with macros, scripts or (custom made) tools.

I often see testers who do a lot of repetitive tasks, for example install a new version of the software every morning, scan log files for errors, create complex reports with standard reports, etc. Why not start up your machine at 6.00am and start automatically an AutoIT script to start the installation, create a tool to scan your logs automatically, or create some SQL statement to export the data to a spreadsheet?

Be lazy, be creative and ask for support from your programmers to automate your boring daily routines. They will love it, writing small tools that solve problems without the need to comply to all the department guidelines.

Use patterns

Almost every programmer knows the 'Gang of Four' and the book they wrote, Design Patterns.

A pattern is a formal way of documenting a solution to a design problem in a particular field of expertise. My definition of a pattern: a predefined described solution for a problem that occurs often.

Programmers use design patterns very often, a pattern that is often used for example is a singleton. A singleton is an object that has only one instance, for example the object that contains all the settings of a user. Many applications have objects that are a singleton. Programmers don't reinvent the wheel every time, they use the singleton pattern.

Why not create test patterns for tests that you have to test frequently? It's not about features you have to test every release. It is about problems that you frequently face, and that can be solved by the same approach every time. For example, testing the log-in functionality or testing creating new data. Read more information about software test patterns on this page http://www.testingreflections.com/node/view/9 .

Write good comments

As programmer I wrote and read a lot of comments in the code. Some example of comments (in pseudo code) you often see are:

```
//If a < 1 then calculate a new value for X
        if a<1 then calculateX;
```

```
try
        CalculateNewDate;
except
        //If an exception occurs during calcu
        //lating of new date we raise an exception
        //raise E.Msg(„Calculation
        //new date has failed");
end;
```

What is the value of these kinds of comments? These are the so-called how-we-do-it comments, adding no value in my opinion. Every programmer can read the statements above and understand what they do. It has no value to describe the statement again in the comments.

A good programmer writes why-do-we-do-this comments. For example in the above code, he describes why "a" should be less than one and why he raises an exception.

As tester always try to write why-do-we-do-this comments, in your test scripts and in your automated test scripts. How you test is clear, describe why you are testing the software.

Conclusion

I like it when there is a healthy rivalry between programmers and testers. It keeps testers and programmers sharp. However, it should really be a healthy rivalry. As a tester you can really learn from programmers! Don't be afraid to ask for their support or advice. A good team has different roles and characters, and it will become even better when they work together as a real team!

Did I miss anything, or did I exaggerate too much... please leave a comment on my blog or via twitter (@raroos). I love to get feedback, it gives me an opportunity to learn!

Sources and background information

Test Automation
http://www.io.com/~wazmo/papers/

Pair Programming
http://www.extremeprogramming.org/rules/pair.html
http://softwaredevelopmentisfun.blogspot.com/2008/07/pair-testing.html

Dojo's
http://softwaredevelopmentisfun.blogspot.com/2010/11/dojo-who-short-introduction-to-dojos.html
http://web.cs.wpi.edu/~gpollice/Dojo.html
http://codingdojo.org/
http://en.wikipedia.org/wiki/Randori
http://en.wikipedia.org/wiki/Kata
http://confluence.agilefinland.com/display/af/Coding+Dojos
http://www.shino.de/2010/06/04/xp2010-testing-dojos/

Be Lazy
http://softwaredevelopmentisfun.blogspot.com/2010/10/automate-your-daily-test-work-with.html
http://softwaredevelopmentisfun.blogspot.com/2010/10/keep-eye-on-your-log.html
http://www.autoitscript.com/site/

Use Patterns
http://en.wikipedia.org/wiki/Design_Patterns_(book)
http://en.wikipedia.org/wiki/Singleton_pattern
http://www.testingreflections.com/node/view/9

> biography

Ralph van Roosmalen

has been working in the software industry since 1997 and has worked with Agile methods since 2005. He has experience as software developer, scrum master, agile coach, tester and test manager. He is currently agile coach, test manager and development manager. Ralph is experienced in Agile software development and integrating testing into Agile projects.
Ralph is a frequent speaker at conferences and writes articles for magazines and his blog. By speaking at conferences and writing on his blog, he is able to interact with other testers, managers, programmers or people with an opinion, thereby improving himself again.
He believes fun and empowering people gives the best chance for successful software projects.
More information can be found on his blog http://softwaredevelopmentisfun.blogspot.com/.

© Katrin Schülke

Improving your Organization's Test processes: Best Practices in the Worst of Times

by Leepa Mohanty

With the increased focus on maximizing returns on IT investments and prudent spending decisions being made in view of the recent financial crisis, improving organizational processes to achieve greater productivity has become increasingly important. The testing organization is also under increased pressure to ensure that waste is eliminated and processes are optimized.

It is in challenging times such as these that a test team's motivation for improvement and optimization of existing test processes can become the key differentiator. This article focuses on best practices that can help ensure increased chances of success, while implementing and institutionalizing a test process improvement program.

It's not a practical expectation from a process improvement program that it will make the process flawless and perfect in all respects. It is more important that the program strives to continually improve the processes making them more efficient than they originally were.

There are two important aspects for continually improving a test process:

- Correcting or fixing problems in existing processes
- Ensuring there are processes that prevent problems from occurring

Losing sight of the second aspect can be very expensive for the organization. It's a known fact that correcting a fault is more expensive in the later stages of the test process. It is therefore crucial that the test team understands that preventive actions are equally important as corrective actions.

The following best practices have been discussed in detail with respect to improving the test processes:

1. *Importance of Self Assessment*

Testing is context dependent and so are test processes. Before embarking on the journey of process improvement, it is important to

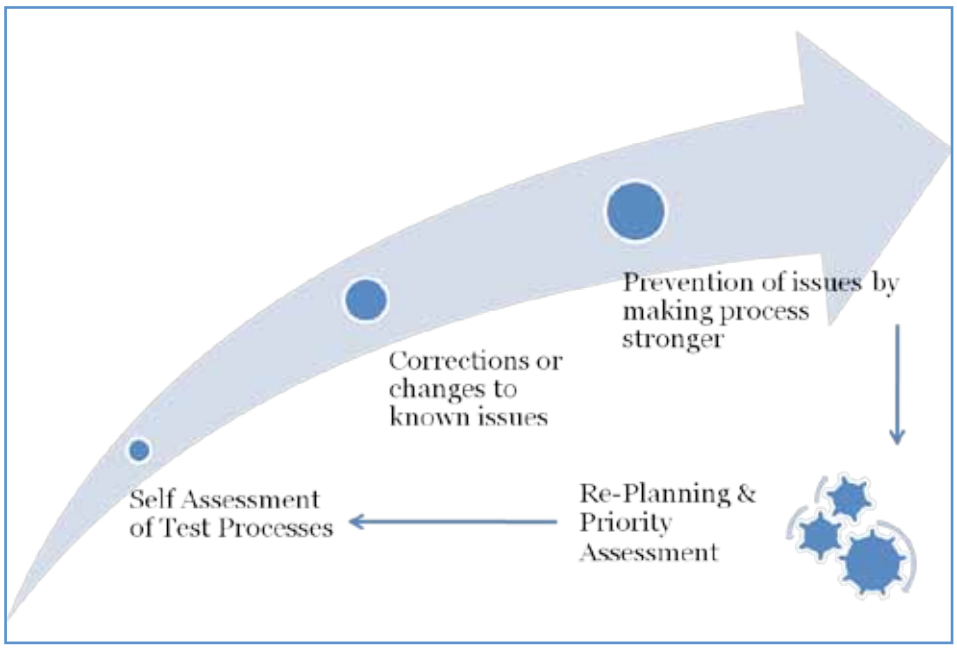


Figure 1 – Continual improvement and self-assessment

make an honest self-assessment of current processes. Knowing the advantages and disadvantages of current processes not only makes the process improvement faster, but also ensures that there is an increased understanding and appreciation, which can go a long way to make the process improvement sustained.

Performing a SWOT (Strengths, Weaknesses, Opportunities and Threats) analysis is a good way to become aware of the internal and external factors that influence the success of the process improvement program.

It is very important that an honest self-assessment takes place before substantial investments are made into the process improvement initiative. Also, it is important to remember that the purpose of SWOT analysis is not to justify or question the reasons of current testing processes, but to analyze and take new decisions based on the identified strengths by aligning them with the opportunities expected to arise in the near future.

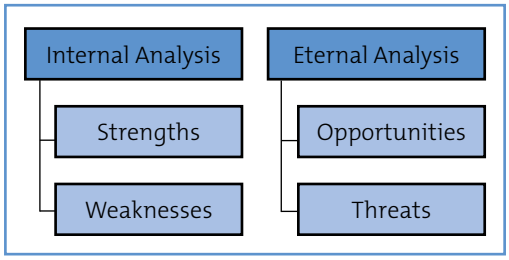


Figure 2 – SWOT analysis

- ***Know what needs to be improved and in what order***

A priority and intended order of activities helps ensure that the time and effort spent on trying to improve your test processes gets a maximum amount of return possible in the intended direction. One approach could be to list all activities performed under each testing phase and assess their relative maturity.

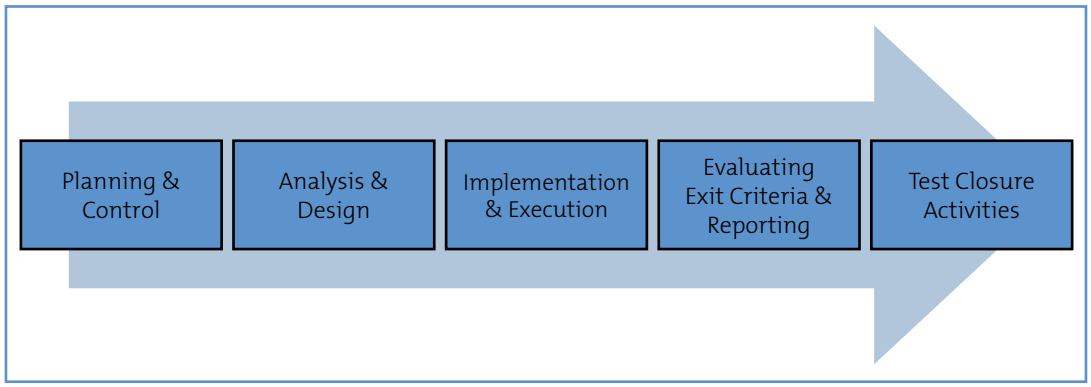


Figure 3 – Testing lifecycle

It is recommended that the top three areas of improvement be identified for each test phase. The list can be refined based on actions taken to fix known issues and the new opportunities, threats and management objectives that become known with time.

Planning & Control	Analysis & Design
1. Test Plan Update Mechanism	1. Traceability Matrix to be implemented
2. Test Policy & Test Strategy	2. Testing could start earlier
3. Exit Criteria Refinement by Test Level	3. Establish criteria for Testeabilty

Implementation & Execution	Evaluate Exit Criteria & Reporting
1. Outsourcing Regression Tests	1. Implement leaner reporting mechanism
2. Cost Saving by Using Tools for Automation	2. Automatic report generation tools
3. Reduce Defect Leakage	3. Quarterly Reviews wih all stakeholders

Test Closure Activities
1. Using lessons learnt to improve process
2. Institutionalizing Best Practices
3. Tracking actions from past meetings

Figure 4 – Improvements by phases

- ***Understanding options available and choosing the best fit***

While a wide variety of process improvement frameworks are available in the industry, it is worthwhile to spend some time in analyzing the options and approaches available.

The models and frameworks available in the industry specifically for Test Process Improvement include:

- STEP
- CTP
- TPI® Next
- TMM

A testing process improvement framework must be chosen keeping in mind your test organization's goals, priorities and maturity. Some of the key aspects of the available models are briefly discussed in this article, which intends to give a brief overview to test managers who are embarking on the journey of test process improvement.

Systematic Test and Evaluation Process (STEP) is a testing process improvement framework that recommends a requirement based testing strategy. The STEP methodology focuses on three important phases of testing and attempts to improve certain process factors (both quantitative and qualitative):

- Planning
- Acquisition
- Measurement

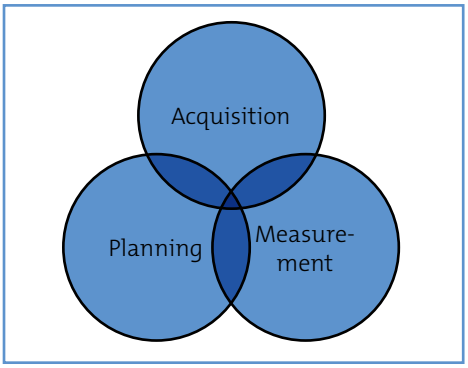


Figure 5 – STEP phases

Critical Testing Process assessment model (CTP) is a test process improvement model that focuses on evaluating the existing test processes that are used in the organization. It is a light weight framework that focuses on critical testing processes instead of all the testing processes and related activities.

A critical testing process can be defined as one that has a direct influence on a test team's ability to impact the quality of the product under test (based on the factors on which testing is performed, like finding and fixing defects in the context of integration testing and building confidence in the application in acceptance testing).

Focusing your test process improvement on critical processes helps ensure that the time and effort spent on process improvement is precisely targeted.

- The first step is the CTP assessment. The assessments vary depending on the context, but the areas listed below are commonly examined:
  - Defect leakage
  - Defect detection percentage
  - Defect rejection rate (tester mistakes)
  - Overheads of testing activities
  - Skills of the test team (domain, technical & testing)
- Both quantitative and qualitative attributes of processes are identified for improvement.
- The strong and weak areas of the existing processes are analyzed and improvement models are suggested for the weak areas.

Test Maturity Model (TMM) is based on the principles of the Capability Maturity Model (CMM) that has been widely accepted and implemented in the software industry. The Test Maturity Model (TMM) provides a framework for assessing the maturity of testing processes in an organization and thus promotes management support and focus on improving test processes.

The five maturity levels are:

- Level 1 - Initial
- Level 2 - Definition
- Level 3 - Integration
- Level 4 - Management & Measurement
- Level 5 - Optimization

It is important to note that each level from level 2 onwards has a defined set of processes and goals. A detailed discussion of the characteristics of each of the above levels is available in various texts related to CMM and TMM models; this is not in the scope of this article.

TPI (Test Process Improvement) is a testing process improvement approach with a focus on business oriented test process improvements. TPI provides a specific, balanced and correlated improvement path for a test organization.

- The TPI model considers the different aspects of the test process, the key areas. The new enhanced model TPI® Next has 16 key areas, including test strategy, stakeholder commitment and tester professionalism.
- In addition to the key areas, the model defines maturity levels and for each of these levels checkpoints, improvement suggestions and enablers.
- A strong focus on ensuring the link between testing process and the business drivers is a key aspect in this framework.

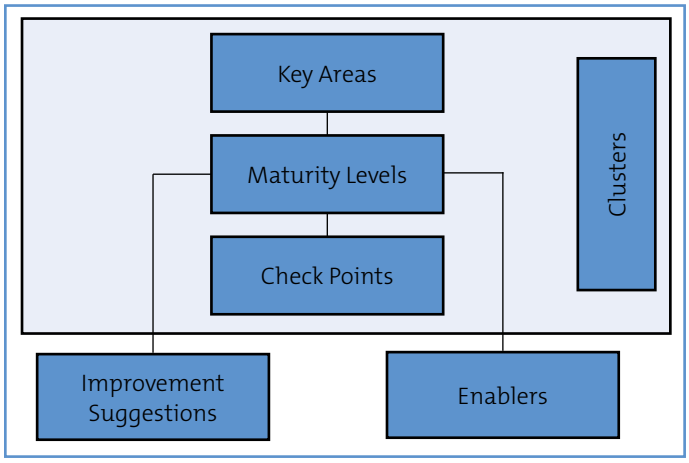


Figure 6 – TPI Next

Choosing an appropriate model based on your organization's goals and priorities is an important first step. It is also a good practice to use a combination of approaches to create an improvement program that is fit for your test organization's purpose.

- ***Collaborating with internal and external departments***

It is important to remember that testing is not an activity being performed for its own sake. The purpose of testing is to support the bigger overall strategy of the business. The goals of testing vary according to test levels and context, but for the most part the influence that internal and external stakeholders have (or should have) on the testing processes is equally important. Collaborating with developers, project managers, marketing team representatives of your organization will provide opportunities to get their inputs for streamlining test processes to best suit your needs.

- ***Planning ahead of schedule***

It is also a good idea to plan ahead of time and build consensus on the objectives, expected outcomes, time commitment and budgetary needs well before starting on the journey of test process improvement.

- ***Benchmarking performance of the testing processes***

Once the test process improvement program has been active for some time, it is good practice to measure the results of the program periodically. Benchmarking performance can give feedback

showing which improvement strategies have worked well and which have not.

- ***Do not change everything at once, plan an incremental rollout***

Rather than adopting a Big Bang approach towards process improvement, it is almost always a good idea to approach process improvement with small steps. The best test managers develop an insight into the relative maturity of the organization's test processes and suggest controllable improvement steps.

Conclusion:

There are plenty of options available to test managers when it comes to implementing a test process improvement program. It is the choice of a particular or a combination of the approaches that is critical for success. The success of the improvement program is ultimately tied to the fit between management commitment, goals and priorities of a test organization and the chosen Test Process Improvement framework. I wish the readers of 'Testing Experience' all the very best in their journey towards implementing improvements in their organization's test process.

> biography

Leepa Mohanty
is an experienced IT consulting professional with 10 years of experience in business analysis, test management and project management. She has successfully managed IT engagements for various clients for Capgemini in USA, Dubai, Singapore and Bahrain. Leepa holds the ISTQB® Advanced Level Certificate in Test Management and is also a certified PMP®. Leepa has extensive experience in the credit card processing domain and is considered an SME in authorizations and transaction processing. In her free time Leepa enjoys making short films.

The High-Velocity Testing Organization

by Nick Jenkins

The Blindspot

Testers specialize in finding other's mistakes, but often have a blind spot when it comes to finding their own. Their skills are honed to identify irregularities in other people's work, but they can be singularly blind to them in their own. Testers also tend towards dogmatic personalities with a black-and-white belief in right and wrong.

Changing any large organization is difficult. Changing an organization which consists of analytical, argumentative, pedantic professionals can be positively excruciating. Testers are often not willing to accept ideas on faith, but must be painstakingly convinced of the 'logical' benefits of a change in method.

How much easier it would be if you could let them convince themselves.

This is the basis of a 'High-Velocity' organization.

The Root Cause

Although we are naturally disposed to be inquisitive, we have done a wonderful job of creating an environment that beats the curiosity out of us. From kindergarten to university to our working life we are taught to respect the authority of 'experts'. [1]

Classrooms, for example, are designed for information transmission, and not exploration; the teacher delivers truth from the blackboard like the sermon from the Mount. And when we go to work, we are told by senior managers, technical gurus and high priced consultants what 'best practice' is and how we should work (CMMi, TMMi, ITIL, etc). Experimentation is discouraged as wasteful and making mistakes is punished by censure or ridicule.

Yet the demand for improvement and efficiency is incessant and urgent. Those that do not evolve are destined for extinction. So, organizations are expected to improve in prodigious leaps as the "next big thing" is delivered by the current crop of experts. This rarely works and is always expensive.

One trend is for 'continuous improvement', empowering teams to continually refine their own processes, raising quality and reducing waste. However, these teams often lack the practical and theoretical skills to identify and tackle problems.

How then do we build continuous improvement into an organization?

The 'Toyota' Way

In the world of engineering there are few organizations larger or more competitive than automotive firms. Amongst them, Toyota is often held up as the shining example of an organization which has risen to the top and continues to stay there (despite some recent challenges). In 2009 Toyota produced more than seven million vehicles, a million more than its nearest rivals [2].

There are many books and articles on 'The Toyota Way', 'Lean' or 'Learning Organisations', and the most convincing of these is "The High-Velocity Edge" by Steven J. Spear.

In it, Spear argues that Toyota (and others) improve not from the products of their philosophy (like kanban, kaizen and jidoka) but from the subtle and pervasive philosophy itself.

Spear's hypothesis is that High-Velocity organizations succeed by finding, confronting and overcoming the limitations that hold them back.

He summarizes it like this[3]:

1. See problems as they occur
2. Swarm and solve problems as they are seen
3. Spread new knowledge
4. Lead by developing capabilities 1, 2 and 3

A High-Velocity organization builds process improvement into every team, every process and every task. This leads to a culture in which self-reflection is encouraged, continuous improvement is ubiquitous and experimentation is rife.

1 *Ken Robinson 2010 Changing Education Paradigm, RSA animation of Ken Robinson 2010 TED talk on Creativity and Education, Royal Society for the encouragement of Arts, Manufactures and Commerce (RSA), viewed 27-Apr-2011, <http://www.thersa.org/events/vision/archive/sir-ken-robinson>*

2 *Multiple authors 2011 Automotive Industry - Automotive production worldwide by manufacturer, Wikipedia, viewed 27-Apr-2011, <http://en.wikipedia.org/wiki/Automotive_industry>*

3 *Spear, Steven J 2009 The High-Velocity Edge, chapter 4, McGraw-Hill, New York*

There are critics of the Toyota Way who point out that much of Toyota's success is driven by its commercial bargaining power, its monolithic culture and the particular devotion of its Japanese workers, not by any particular management science[4].

However, Toyota is just one example and the principles of High-Velocity or learning organizations resonate with many managers. Similar themes can be found in books like Peter Senge's 'The Fifth Discipline'.

Seeing Problems As They Occur

Testers specialize in seeing problems after the fact – finding defects by examining software for the traces left by bugs. We are deductive professionals par excellence.

What we're not very good at is seeing problems **when they happen**.

There is power in observing problems directly in the context of process, place and time; the more distance in each, the more contextual information is lost. Rebuilding that context is a classic example of waste. If we can identify problems when they happen, we can isolate the root cause from the symptoms easily.

We can then resolve or counter the problem at its root.

For a long time people have pointed out that solving problems earlier in the life cycle is cheaper (Barry Boehm) and that testing should move further up the lifecycle (TMMi et al). Normally this is characterized by breaking a waterfall process into some kind of incremental review where errors are still detected after the fact, albeit earlier in the lifecycle (V-model etc).

In a High-Velocity organization, problems are assessed in the moment they are detected. In the context of a Toyota factory, this is typified by 'andon' cords attached to work stations. When a cord is pulled, the assembly line stops, a red light signals the location of the problem and engineers assemble there.

Swarming the Problem (and Solving it)

In traditional organizations, leaders are left to deal with problems alone. In a learning organization, an individual that discovers a problem calls on colleagues to diagnose the cause, select options and predict outcomes.

The team 'swarms' the problem to bring its collective mind to bear.

Rather than identify a single option, the method emphasizes generating multiple options (hypotheses), predicting the outcomes and rapidly testing the options to select the best solution (experiment). Experimenting supplies this evidence and avoids blind alleys and false starts; it also exposes assumptions and constraints for critical examination.

The role of observers is not to offer solutions but to critique the problem holder's method – to sharpen their problem solving skills. This avoids the trap of expert authority dominating the solution. The purpose is to refine the group's collective problem solving skills and not necessarily to find the 'perfect' solution to the problem.

Give a man a fish; you have fed him for today. Teach a man to fish; and you have fed him for a lifetime.

Share the Knowledge

Swarming a problem leads to the sharing of knowledge as a circle of colleagues discusses options and techniques. This reinforces the continuous nature of personal improvement in a learning organization and leverages existing knowledge.

However, it's also important to share the knowledge outside of the immediate circle. All too often what is learnt in one part of the organization will be needed in another, as the same problems are faced by many teams.

An organization must have a way to communicate what is new across its length and breadth. Quality circles, knowledge bases or documented quality improvements can all support knowledge sharing, but the most powerful methods involve face to face sharing across teams.

This does happen in contemporary organizations, but occasions are limited and the focus is often poor. Teams are usually preoccupied with saving face or creating a good impression. Rarely, if ever, do teams come together for a critical and reasoned debate.

To support the learning organization, it is important to share not only problem and solution but context, reasoning, hypotheses, experiments and outcomes. This builds the corporate memory of the organization and inhibits reinventing the wheel.

Lead by Teaching 1, 2 & 3

Spear emphasizes the role of leaders in coaching the methods of problem solving. This is the heart of what he sees as a High-Velocity organization. A leader doesn't lead by command and control but by mentoring in problem resolution.

In one example from a Toyota plant, a senior manager inducts a new senior manager by telling him, 'Everyone knows you're the boss, but I want you to manage as if you had no power over your subordinates.' The VP of Engineering would have to 'sell' his ideas on the shop floor[5].

In a learning organization, the manager is not the expert source of knowledge, they are the facilitator of a learning process in which the subordinates develop their skills. When it comes to a problem, the leader's opinion is no more authoritative than the subordinate's. It is in the technique of problem solving that he or she excels.

This resonates positively with the challenges many organizations find training and retaining qualified employees. Employees often leave employment not for reasons of pay or conditions but for the lack of challenge and opportunity in the job. A learning organization tackles both the organisation's need for constant improvement and the individual's need for constant challenge.

The High-Velocity Testing Organisation

So how does this apply to software testing?

To truly contribute to software development, testers must move back up the value chain of the software development lifecycle

4 *Nomura Masai, AMPO Japan-Asia Quarterly Review, Vol 25 No .1*

5 *Spear, Steven J 2009 The High-Velocity Edge, p293, McGraw-Hill, New York*

(SDLC). Relegated to the end of the SDLC, they seek mainly to limit harm, not add value.

In the context of traditional waterfall, testers struggle to become involved earlier because of their deductive nature – they are always waiting for an artefact to be delivered so they can 'test' it.

As part of a High-Velocity organization, testers can apply their deductive skills to finding and diagnosing the root cause of problems as they occur. They can be an integral part of the team, helping developers test options and critically assessing assumptions and predictions. They can help see the wood for the trees.

Imagine a software development effort which stops every time a bug is detected and the root cause is discovered, diagnosed and countered on the spot.

To some this might appear wasteful, chopping and changing tasks, but developing software is not factory work – it's a creative discipline. In creating something, you can choose to continue propagating the same error repeatedly, or you can stop, remove the source of the error and continue free from blemish. One has the hallmarks of a craftsman, the other smacks of rank amateurism.

Parallel Ideas

The concepts espoused here have parallels with contemporary software testing movements such as Context Driven Testing and Agile.

Context Driven Testing emphasizes the importance of selecting the right practice for the right task and deprecates generic "best practice". It emphasises that individuals examine the problem at hand and, using the techniques at their disposal, select the best approach to the problem[6].

Context Driven Testing also espouses the personal development of a tester as a domain expert with a wide selection of techniques at their disposal. They become a generic (test) problem solver and not an advocate of any single practice.

Agile has similar principles with flexible processes and an emphasis on individuals and interactions over processes and tools. Agile practices like standups and retrospectives have their roots in the group based problem solving of Lean & The Toyota Way .[7]

Agile also supports the notion of finding and fixing defects as they occur through techniques such as Test Driven Development (TDD) and Behavioral Driven Development (BDD). Agile teams also often sport andon cords in the form of 'build lights' which alert the team immediately to the fact a build has failed.

Conclusion

Change is hard.

We are creatures of habit and the warm safety of our occupational rut is often far too comfortable. The strength of the ideas behind learning organizations enables people to change their world in incremental steps, not Big-Bang chunks.

As one manager put it to Peter Senge [8], "People like to change. They don't like to be changed."

There are plenty of options in testing at the moment : TPI, CAST/ISEB/ISTQB, automation vs manual, Agile techniques, exploratory testing, TMMi/CMMi, context driven, cleanroom, formal methods, etc.

The ideas behind Learning and High-Velocity Organizations offer a philosophy with which to bind all of these together. By focussing on personal decision making and experimentation you encourage teams to choose their own destiny.

Instead of force-feeding your testers the next evolution of 'best practice', you can lay out a smorgasbord of options, confident that they will have a process for selecting the right tool for the right job.

The 'problem' focus of a High-Velocity organization also ensures that effort is well directed. Every problem removed from your process increases long term quality and reduces waste.

Yet, these techniques offer no silver bullet.

Breaking down the problem into stages helps, but like the generic process problem, you must observe the problem of change in your organization. What does "seeing the problem" mean in your context? How will you "swarm" problems? How can you "share new knowledge" with colleagues?

There is no doubt that the journey is long and painful when taken in small steps; there are many barriers to success. However, the lofty ambition of an organization which delivers continuous improvement along with personal development for all its employees is one worth striving for.

> biography

Nick Jenkins
started his testing career in Sydney in 1996 when he helped establish the Australian Testing Centre, the first independent test house in Australia. He also trained with James Bach at STLabs in Seattle.

Since then he has worked in London, Prague, Boston and San Fransisco for telcos, off-the-shelf software companies and in the finance sector. Currently he resides in his home town of Perth, Western Australia where he works as head of Quality and Logistics for Bankwest, a national bank.

He spends his spare time trying to keep up with his 5-month old son Henry and frolicking with kangaroos in the bush.

6 *Cem Kaner / James Bach 2010 The Seven Basic Principles of the Context-Driven School – viewed 27-Apr-2011 <http://www.context-driven-testing.com/>*

7 *Various The Agile Manifesto – viewed 27-Apr-2011 <http://agile-manifesto.org/>*

8 *Peter M Senge 1994 The Fifth Discipline, Doubleday*

Cloud Computing: A Leap Forward In Improving the Testing Process

by Ashish Kriplani

The growth of Cloud-based computing is outstripping even the most optimistic predictions. That growth is based on a compelling value proposition: speed to market, agility to bring forward or retire services, and the chance to move from capital expenditure into operational expenditure. How does this affect testing? Can the use of Cloud computing help to reduce the costs or complexity of testing or speed up testing?

Cloud computing refers to the provision of computational resources on demand via a computer network, such as applications, databases, file services, email, etc. In the traditional model of computing, both data and software are fully contained on the user's computer; in Cloud computing, the user's computer may contain almost no software or data (perhaps a minimal operating system and web browser only), serving as little more than a display terminal for processes occurring on a network of computers far away. A common term for a provided Cloud computing service (or even an aggregation of all existing Cloud services) is simply „The Cloud".

This article looks at how testing can be revolutionized through the evolving Testing Cloud models, outlining the potential benefits and evaluating the challenges, the services being developed and providing an outlook for Cloud Testing. That growth is based on the compelling value proposition of delivering business productivity in the Cloud: speed to market, agility to bring forward or retire services, and the chance to move from capital expenditure into operational expenditure. Although Cloud computing is still in its infancy, it is becoming increasingly clear that the Cloud model will supplement, if not entirely replace, mainframe and client/server installations in the years to come.

1. *Testing On The Cloud – Rapid, Cost Effective And Growing.*

As Cloud computing evolves, and Cloud service adoption becomes ever more wide-ranging, a new global infrastructure is being created; this infrastructure can easily be connected to traditional infrastructure (including legacy systems), as shown in Figure 2. However, it is not just for business IT assets that Cloud computing reduces previous limitations. It does the same from a software or application testing perspective, removing the typical constraints presented by having to test on client-owned or internal resources.

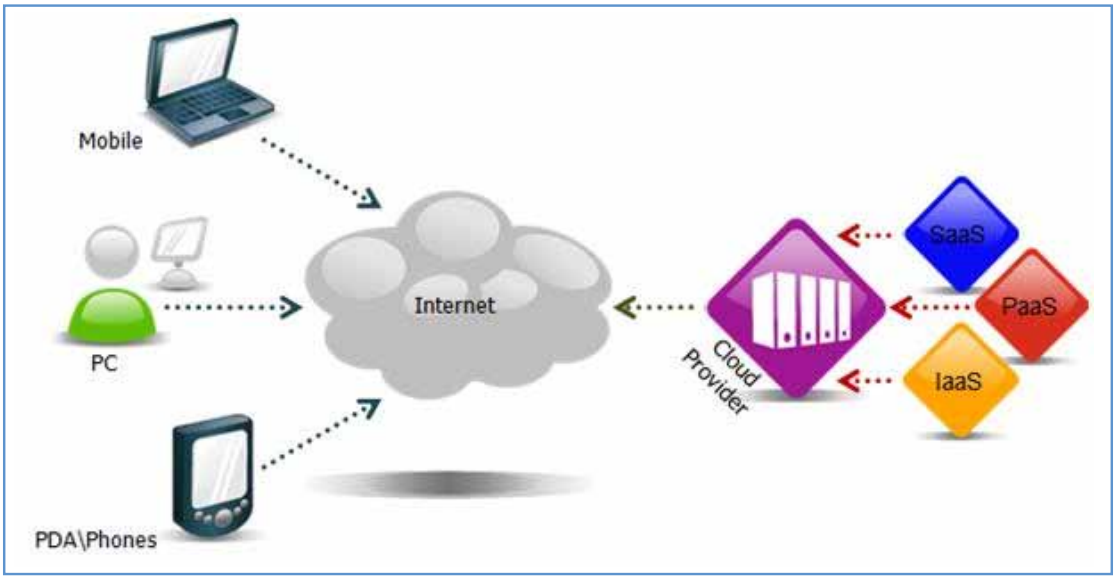


Fig: Cloud Computing Scenario

A Cloud infrastructure creates significant new opportunities for software quality assurance and testing.

In our view, Cloud computing is a model or platform on which testing can be carried out just like any other service. It enables convenient, on-demand network access to a shared pool of configurable computing resources. Those resources, from networks, servers, storage, applications and services, can be rapidly provisioned and released – thereby drastically reducing management effort and service provider interaction.

2. Test Environments Delivery Scenarios

We have outlined below possible scenarios for organizations whenchoosing the most appropriate test environment:

Private IT Vendor Cloud for Managed Testing Services:

With a wide range of services necessary in a Managed Testing Service, the capabilities of this infrastructure need to be flexible and scalable whether on or off-premise, or whether global or local (according to data management regulations). Clouds can be created with their own server parks, in which the server is divided between the different projects when those projects need the capacity, creating a more flexible server organization. The client's (test) data will reside outside its premises, on the test Cloud.

Private or Hybrid Cloud at Company's own locations:

Using existing owned infrastructure and maintaining a high level of control by keeping it on-site is a relatively risk-averse option. Keeping test data and applications on-site, it offers more than test infrastructure, opening up other Cloud computing options.

Hybrid Cloud offered by IT providers and serviced by IT vendors:

Outside the client's firewall, clients use the IaaS from a Partner company to build Development and Test Clouds available only for them, which are then managed by IT vendors for the client. Such Clouds offer flexibility to increase or decrease capacity as required.

3. Test Tooling From The Cloud

SaaS offers great value in using test tools from a Cloud environment, recognizing the dilemma of cost and license commitment that many organizations face when wanting to use test tools. Test tooling from the Cloud provides a quick and efficient service-based solution. Test teams can purchase 'on demand' (or by subscription model) the download and subsequent use of these tools, hosted on the Cloud, as and when they are required for a specific project or program.

Whenever a new project needs to employ specific test tools for a short to medium amount of time, Cloud-based test tools can be used, see Figure 4.

In addition to the benefits of greater flexibility in tool usage, there is also minimal investment or reduction in license fees, and the elimination of ongoing tool maintenance further increases the cost savings.

In this way, the cost of tools is reduced because it is better aligned with actual usage, availability is immediate, and by using the tool consistently over a group of projects, increased standardization and a quality management ethos is embedded.

4. Key Benefits Of Testing On The Cloud

All levels of testing can be carried out in Cloud environments, and indeed, some types of testing benefit particularly from a test environment in the Cloud.

4.1. Flexibility

- Different levels or grades of tests can be executed on separate environments at an organization's convenience;
- Testers no longer need to wait until the end of the testing phase to move to a 'production-like' environment for their performance, load and stress tests. Instead, a production-like environment can be brought into action at will.

4.2. Simplicity

- End-to-end testing can be set-up in the Cloud relatively simply and quickly, provided the necessary servers and images can be accessed to create an end-to-end environment;
- The Cloud also offers a new level of simplicity in terms of training or bug-fixing environments, which can be launched as quickly as the images and configuration can be put in place.

4.3. Comprehensive and indicative testing

- Even end-to-end tests for more generic business processes can be carried out in the Cloud. All the necessary components can be published in the Cloud to create the whole chain of systems. In this way, the whole business processes can also be tested;
- In the Cloud, a more 'realistic' load can be generated than the virtual load generated by other tools. Cloud-enabled performance test tools generate the needed load and stress to test an application more accurately.

4.4. Cost reduction

- There is a reduced need for expensive environments, which need to be used only when tests have to be executed;
- Historically, internal testing and acceptance environments have been permanently available for testing projects within a company, creating a permanent pressure on development budgets and infrastructure resources. Cloud environments, however, can be enabled and disabled at will, reducing the cost of environment management.

4.5. Cleaner, greener testing

It is intuitively true that the efficiencies Cloud computing provides make it significantly 'greener' than traditional, hosted models. That is true for testing, too. By sharing and centralizing Cloud resources for their test infrastructure, businesses will use IT resources 'on demand' and eliminate waste. In addition, clients using Cloud data centers can minimize energy use and deliver environmental savings in CO_2 of up to 55% .

4.6. Driving Standardization

In one sense, creating test environments in the Cloud is merely a temporary first step on a much longer journey. Cloud computing will give a huge impetus for the standardization of organizations' infrastructure. That key shift will inevitably catalyze IT modernization and improve the maturity of internal IT services, impact upon application consolidation and portfolio rationalization, and change the way enterprises look at resourcing IT services internally.

Fig: Cloud Computing Features

5. Conclusion

The IT industry faces significant changes that will impact on most areas of the traditional IT department, including the software development lifecycle. In years to come, more and more organizations will implement their development and test environments in a Cloud environment, and Agile performance testing will become mainstream.

There are many opportunities brought about by Cloud computing; some are more evolutionary (such as private and public test Cloud building, testing Cloud-based applications, integration services, and the use of SaaS test tools), while others are more revolutionary (such as brokering and offering test Cloud services). At the heart is consolidation – environment and application consolidation and portfolio rationalization as a means to lower the cost of operation.

The real opportunity to cut costs of test environments in private, hybrid, public or community Clouds lies in both their management and maintenance. When Cloud resources are not needed, they can be turned ‚off' cheaply and efficiently, and then 'on' again as necessary, without implementing the environment again. Cloud computing delivers on-demand testing and acceptance environments, reduces complexity and even can speed up the testing process.

> biography

Ashish Kriplani
Software Engineer in Capgemini's Financial Services group. Working in HSBC accounts for the past nine months in Mainframe testing.

Optimizing the Testing Procedure by Checkpoints Convolution

by Elena Sviridenko & Alexey Ignatenko

It's a well-known fact that the testing process takes a lot of time and requires extra accuracy and attentiveness. Extra time, extra effort and extra resources cost much, and if it is possible to reduce all of these, it is worth doing so. In this article we would like to share some practical experience and show what can be done to spend less time/effort/resources (whatever you choose) on the testing cycle. The main idea of the approach is to reduce the total number of test steps, while still covering all the requirements. The approach consists of the following phases.

Requirements breakdown. Let's talk about functional testing that should be performed on every single project. The more requirements are covered by test cases, the better the quality that is achieved. So we need to have the defined and approved requirements as an input. They can be in any format, but for our purposes, it is required that they are broken down and numerated. Thus, the first thing you should do is to atomize the requirements: break them into smaller pieces, each of those can be covered by a few checkpoints. Let's consider the following example. Requirement states: "Button A should be enabled only when textbox B is not empty". This requirement is broken down into two parts:

1. Button A should be enabled when textbox B is not empty.
2. Button A should be disabled when textbox B is empty.

Part 1, in its turn, is split into two checkpoints:

1. Button A is enabled when textbox B is not empty.
2. Button A becomes enabled once anything is typed into textbox B.

Whereas part 2 is split into similar checkpoints:

1. Button A is disabled when textbox B is empty.
2. Button A becomes disabled once textbox B becomes empty.

Let us call the set of requirements R, and the set of checkpoints CP. There is no need in the requirements traceability matrix per se (mostly on time saving grounds), but we should know what requirements are represented by what checkpoints, so in case of a change request we can easily pinpoint all the requirements and checkpoints affected by it. (Later we will show the solution for this.) Once we got all the requirements covered by the checkpoints, we need to define the set of software system states in which those checkpoints can be verified.

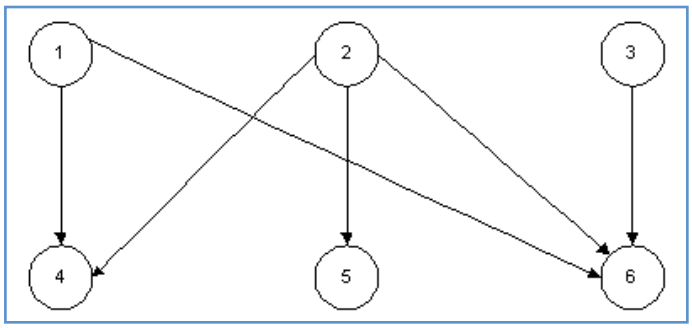
Defining system states. The set of states is system specific. You need to have a strong understanding of the system flows to define what states are actually important for testing. Software systems can have from one to many initial states and from zero to many terminal states. The initial state is the state from which the software system starts to work. It is defined by configuration parameters. As soon as the parameters values can vary, there is a non-empty set of initial states, let's call it S_o. The terminal state is the state from which it is impossible to go to any other state. Some systems (for example, services) do not have such states; others can have one or a set of terminal states, let's call it S_{term}. All other system states (those that are neither initial nor terminal) are included in the set of states S. To get to a particular state, a software user should perform a set of actions, whereby each of those actions transfers the system from one state to another. Being in state S_i allows us to verify from zero to many checkpoints. Let's call the set of checkpoints that can be verified in state S_i as $CP(S_i)$. On the other hand, every checkpoint CP_j can be verified in a set of states, let's call these $S(CP_j)$. The main testing goal is to check all the checkpoints from set CP for the minimum number of actions, i.e. for the minimum number of transitions between states. Generally speaking, the problem of covering the set of states with a minimum set of transitions is NP-complete. And its optimal solution can be found by exhaustive search only. In practice, an optimal solution is not required, because finding it will take much more time than would be saved for testing. We will show some basic optimization principles that can help finding a solution that is close to optimal.

Checkpoints convolution. The following principles can significantly reduce the time needed for checkpoints verification.

1. Terminal states with only one route can be excluded from the further optimization process. From the set of checkpoints *CP*, select those that can be checked in a single terminal state only. Create a set of such checkpoints called CP_{term}. Obviously, all states from set $S(CP_{term})$ must be reached to assume all checkpoints have been tested. So select those states from that can be reached by one route only. Call the selected set

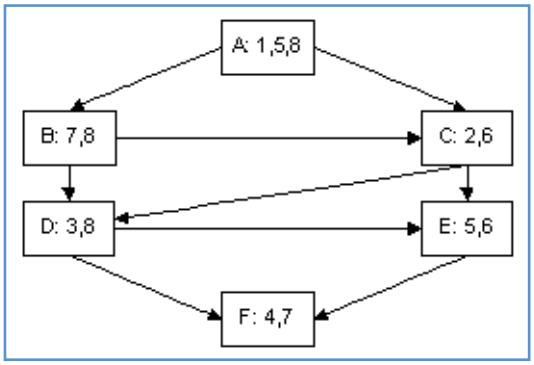
of states $S(CP_{term})^{\{i\}}$. All checkpoints that are checked in states between S_o and $S(CP_{term})^{\{i\}}$ can be marked as covered and excluded from the set CP further on.

2. All the checkpoints, which should be checked in non-initial states, must be combined with checkpoints that should be checked in initial states. For example, we have six checkpoints ($CP_1 - CP_6$) and six states ($S_1 - S_6$). Each checkpoint CP_j can be checked in state S_j only. States $S_1 - S_6$ are initial states, whereas states $S_4 - S_6$ are non-initial states. The system's state diagram is shown in Figure 1.



In this example, the principle forbids verifying checkpoint CP_6 in one test sequence with either CP_1 or CP_2 because such a test sequence will leave CP_3 not combined with any checkpoint from a non-initial state, and this will lead to extra testing effort.

3. All checkpoints that can be checked in many states should be combined with checkpoints that can be checked in single state only. Find those checkpoints that can be checked in one state only. The set of such checkpoints will be called $CP^{\{1\}}$. Similarly, the set of checkpoints that can be checked in two states is called $CP^{\{2\}}$. The set of checkpoints that can be checked in the maximum number of states is called $CP^{\{N\}}$. On the other hand, the set of states where checkpoints from $CP^{\{i\}}$ can be verified is called $S(CP^{\{N\}})$, and so on. The convolution process starts from set $CP^{\{N\}}$. For each of the checkpoints from that set, find out if it can be verified in any state from $CP^{\{i\}}$. If yes, this checkpoint is marked as covered. If not, find out if it can be verified in any state from $S(CP^{\{2\}})$. If yes, states from $S(CP^{\{2\}})$ in which the checkpoint can be verified acquire additional weight that will be used in route selection. The more weight a state has, the higher the probability that it will be visited during testing. States from $S(CP^{\{1\}})$ have infinitely large weight which means that they have to be visited. When all checkpoints from $CP^{\{N\}}$ are combined with sets of lower order, the convolution process is performed for checkpoints from $CP^{\{N-1\}}$, and so forth. Finally, we obtain a minimized set of states (with their weights) that can cover verification of all checkpoints. Let's consider a simplified example depicted in Figure 2.



There are 8 checkpoints (from 1 to 8) and 6 steps (from A to F). $CP^{\{1\}}$ = {1, 2, 3, 4}, $CP^{\{2\}}$ = {5, 6, 7}, $CP^{\{3\}} = CP^{\{N\}}$ = {8}; $S(CP^{\{1\}})$ = {A, C, D, F}, $S(CP^{\{2\}})$ = {A, B, C, E, F}, $S(CP^{\{3\}})$ = {A, B, E}. The convolution process shows that all checkpoints from $CP^{\{2\}}$ and $CP^{\{3\}}$ can also be verified in states from $S(CP^{\{1\}})$ so visiting those states can cover verification of all checkpoints. The optimal route is obvious: A -> C -> D -> F, so all 8 checkpoints can be verified by performing only 4 steps.

Checkpoints representation format. Below in Table 1 we propose a format for defining checkpoints. This example is based on the states and checkpoints from Figure 2.

CP#	R#	Check Points	Actual Result
State		<state summary> *(e.g., image is loaded)*	
Steps to get to a state, *e.g.*: *1. Load a JPEG image*			
1	1	<check point #1 statement> *(e.g., button "Save" disabled)*	passed
5		<check point #5 statement>	failed
8	2	<check point #8 statement>	passed
State		<state summary> *(e.g., State A + press "Rotate Left" button)*	
Steps to get to a state, *e.g.*: *1. Enter State A* *2. Press "Rotate Left" button*			
2	3	<check point #2 statement>	passed
6		<check point #6 statement>	passed
State D		<state summary> *(e.g., State A + press "Rotate Right" button)*	
Steps to get to a state, *e.g.*: *1. Enter State C* *2. Press "Rotate Right" button*			
3	4	<check point #3 statement>	passed
State		<state summary> *(e.g., State D + press "Save" button)*	
Steps to get to a state, *e.g.*: *1. Enter State D* *2. Press "Save" button*			
4	6	<check point #4 statement>	passed
7		<check point #7 statement>	failed

where CP# – a sequence number of a checkpoint;

R# – a sequence number of a requirement;

Check Points – the column that contains the checkpoint statements;

Actual Result – the column that contains an indication whether checkpoint failed or passed;

State X – a sequence number of a state;

<state summary> – a short description of a state.

The format offered in the Table 1 can easily be used with Microsoft Excel. For instance, you can have three spreadsheets in Excel workbook:

- Revision history (to track the changes);
- Requirements (the numbered list of the requirements already broken down);
- Checkpoints (the same content as in Table 1).

This format has two main benefits: we have requirements traceability "out of the box", and some important testing metrics can be precisely calculated without any additional effort. Such metrics are DOF ("degree of freedom") and DOF index (find more details about DOF in [1]). Based on the fact that all the checkpoints are atomic, DOF would be equal to the total number of checkpoints. So there is no need to spend additional time on analyzing the expected results of a test case to derive all the possible fails. For the case shown in Table 1, DOF would be equal to 8 (the number of checkpoints), whereas the DOF index (a ratio of the failed checkpoints to the total number of checkpoints) would be 2/8 = 25%.

Summary. The described technique is very beneficial especially for regression testing. This is due to the following two reasons:

- We assume that checkpoints and states obey the Pareto law [2], i.e. 20% of states can cover verification of 80% checkpoints.
- Residual 20% of checkpoints (that require 80% of states to be visited and cannot be easily combined with other checkpoints) can be excluded from regression testing because the probability of finding defects during their verification is low.

In conclusion, let's summarize the main points of described approach:

1. Break down the requirements and numerate them.
2. Create a set of atomic checkpoints to cover all the requirements and numerate them.
3. Define a set of testing states to cover all the checkpoints.
4. Perform checkpoints convolution and find routes that can verify all checkpoints with the minimal number of steps.

References.

1. Alexey Ignatenko. Predicting number of defects by means of metrics // Testing Experience. – 11. – 2010. – P. 96 – 97.
2. http://en.wikipedia.org/wiki/Pareto_principle

> biography

Elena Sviridenko

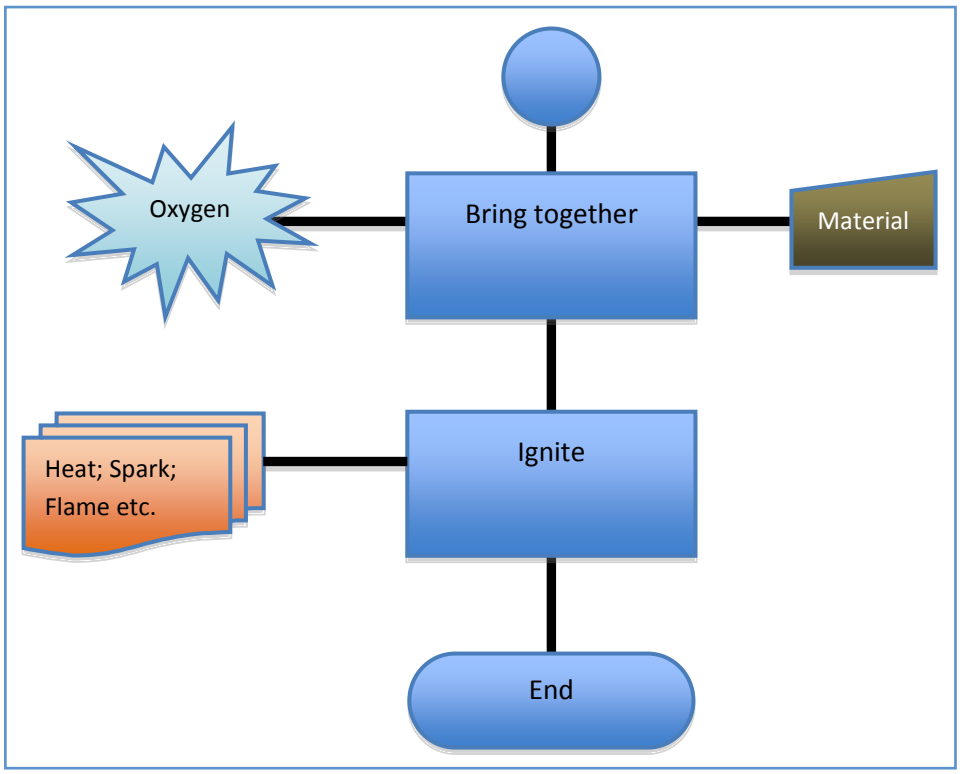
QA Engineer at TEAM International, entered Kharkiv National University of Radio Electronics in 2002 and graduated Magna Cum Laude with a Master degree in computer systems and networks in 2007. She started a QA career in 2004 at Direct EDI, Inc, where she later acquired the QA lead position. In 2009, she started working for Provide Support LLC as a customer service support operator, and after a year switched to QA engineer there. In March 2011, she joined TEAM International as a mid-level QA engineer. There she was involved into various projects, including client-server and desktop applications on different Windows and Linux platforms, EDI-related applications, web analytics and website monitoring systems, content management systems. Depending on the position, the responsibilities included business analysis, requirements analysis, creating the project accompanying documentation (specifications, test documents, user guides, help manuals), software testing, junior QA training, interviews, technical and customer support.

Alexey Ignatenko

Lead QA Engineer at TEAM International, entered the National Technical University in Kharkov, Ukraine in 1998. In 2002, he earned a bachelor degree in computer science and in 2004 a master degree in system analysis and research. After that he started a postgraduate course on progressive information technologies and in 2006 earned another master degree in intellectual property. He started his career in 2004 as QA engineer at Mega Business Software Ltd., and later acquired the position of configuration manager. He also started lecturing part-time at the department of automated management systems at the National Technical University. In 2006, he started to work for DataSoft company as a technical support engineer. In December 2007, he was employed by DataArt Solutions Inc. as QA manager, where he later became configuration manager and project manager. At the beginning of 2010, he was employed by TEAM International as lead QA engineer, which is where he works now. He was engaged in lots of projects, including games, recognition systems, CRM systems, data providers, content management systems, incident management systems, etc. He has had a wide range of responsibilities, such as requirements analysis, software testing, document review, configuration management, customer support, quality assurance, etc.

When does change become improvement?

by Thomas Hijl



This question has kept me busy for weeks, ever since I saw the call for articles: "Improving the test process.". Improving a ***process result*** – a product forx example – is easier to measure, in my opinion. When more people buy it or like it better than the older version, it probably has improved. However, for a process it is a bit more complicated, I think. How do you measure whether a process has improved? When does change become improvement?

Last week the weather in The Netherlands was beautiful, and at weekend evenings I always like to make a small fire outside. I pour myself a nice glass of red wine and gaze at the stars. By the way, there was a clear and magnificent shiny falling star that shot through the sky, ... yes,... I made a wish. Anyway,.....

While staring at the stars and the warming flames in the basket, I wondered whether the process of making fire has improved over the last ten thousands of years. Or have we just changed the process to fit our new way of living. At the basis, making a fire is a very simple process:

1. Make sure there is oxygen
2. Gather material that can burn
3. Ignite the material
   a. by heating
      i. by spark
      ii. by flame
      iii. other

What is there to improve, I asked myself. I tried hard to find a process in my daily life - other than test processes ;-) – that has been improved and that I could use to make a comparison with test processes. I simply could not come up with anything. There have been loads of changes to processes, but who is to judge whether a process has improved? On the other hand, there are of course processes that have been improved. A classical dilemma.

I gave up searching and had to come to a conclusion to ease my mind: "Whether a process has improved or not is dependent on a subjective goal.". Was the goal to make the process quicker, more efficient, more effective, cheaper, ...? The specific goal has been set by someone and is therefore subjective. Another person might have a different goal with the same process and accordingly conclude that the process has deteriorated through the change that was made to it. This is all I could come up with about process improvement.

So, ... this is the bottom line: I have found and am still finding it very difficult to write something in-depth about process improvement, which makes me very curious to read this quarter's Testing Experience and see what others can teach me in this respect.

> biography

Thomas Hijl
is acceptance manager and partner at Qwince BV. He started in 1996 as consultant / project manager for Philips Semiconductors (a.k.a. NXP) working on technology projects in the EMEA, APAC and AMEC regions. He switched to test management because of his ambition to improve project deliverables. He now serves clients with managing acceptance throughout the project lifecycle.

Building Lasting Improvement from the Bottom Up Using a Top Down Approach

by Thomas P. Wise

Ward and Peppard (2002) speak of the IT delivery process as a comprehensive program of analysis of the environment, building toward a set of business requirements, and ending with a planned delivery strategy that complements the strengths of the organization, and strengthens the weaknesses. In the year 2010 Comcast Cable began an evaluation of our engineering delivery strategy with the goal of strengthening the division's ability to deliver powerful customer opportunities for a comprehensive and participative experience that complements the active lifestyle of media users in the 21st century.

In a world of interdependent economies and international world events that close the miles that separate us to only milliseconds of geographic separation, our world is dependent upon the effectiveness of software systems, therefore the effective practice of the quality profession is essential (Kruchten, 2000). Understanding that embedded within every IT delivery process is a basic assumption regarding quality processes, Comcast Cable's National Engineering and Technical Operations (NE&TO) systems integration test organization determined to remake itself as an industry leader in quality through participative and continuous process improvement.

Context

Systems are best when the design is driven by the organizational strategy with enough depth to understand and incorporate the desired quality, rather than delegating the desired level of quality to the project team (Peskin && Hart, 1996; Ward & Peppard, 2002). Collaborative processing is essential in mission critical structures, and when undertaking an organizational facelift in mission critical line organizations, the effective use of the many available tools is essential to the outcome (Grunbacher, Halling, Biffel, Kitapci, & Boehm, 2004). With an extensive background in international IT process consulting, Adrian O'Leary, Vice President of the NE&TO National Lab Quality Assurance Department (CQA) in Downingtown, Pennsylvania, canvassed his organizational peers in engineering and operations regarding customer experiences and cross organizational relationships. Armed with this information, O'Leary set CQA on a path of self-improvement.

Process

Improvement begins with determination of an end goal and, with the goal in sight, selection of an improvement model that fits the organization. NE&TO goals include the ability to identify specific practices and experiences with which projects are to be accomplished based on the desired outcomes of our internal customers. Resources may be allocated based on the desired customer experience. Basili, Caldiera, and Rombach (2002) described this model as a software factory with predetermined capabilities, practices, procedures, and outcomes, using prescribed methodologies and methods to ensure an effective and predictable quality outcome every time. With this goal the CQA organization set out to identify an effective improvement model to drive toward the factory as an end goal.

Many models exist including CMM, COBIT, ITIL, ISO, and the list goes on-and-on to include many proprietary tools. Models contain descriptions of organizational maturity generally beginning with expressions of heroism, progressing toward documentation of practices, procedures, processes, and moving into effective measures and metrics with reporting that guide the organization in targeted and controlled improvement. Each model has, as an essential part, a description of maturity practices that define the organizational capability to function, thus defining the ability to consistently deliver. A key to succeeding in any self-improvement program is to ensure the program fits the character of the organization. With this in mind, and heeding the warnings of Sterman, Kofman, and Repenning (1997) regarding the trade-offs of TQM programs and bottom line stresses created by overzealous improvement programs, and with the cable industry's history of self-determination and independence, it was decided to use a generalized model with little prescription guiding the elevation of maturity.

Assessment Methodology

Assessments, like solid research, must begin with the end in mind. Desired methods of analysis must drive the type of information collected and the means of collection. An assessment core team was formed using a combination of employees and management "volunteers" to guide and formulate the assessment process and tools based on the capabilities and norms of the department. A research methodology was chosen, and the development of methods to be used was begun.

With constraints regarding time and project needs, in order to simplify and shorten the data collection and analysis process, the team chose to stick with simple quantitative methods. This means that the entire process must be focused on quantifiable, empirical, or observable, data establishing the constraints regarding available collection methods. Team members were trained in data basics and collection methods in a couple one-hour sessions that included information on the data types, potential collection methods, and the preparation of research questions.

Determining the Collection Methods

Every research, or internal assessment, should begin with a statement of purpose. Our purpose, like most development and engineering test organizations, was to support more development in less time, and to free up resources moving CQA toward the desired factory model. Our purpose was simple, clear, and unambiguous. In order to establish a useful assessment model, we set forth to provide clarity and guidance in the process.

A statement of the management dilemma was agreed by the core team members. "How can an organization grow from a service model to a factory model?" This statement provided guidance to the assessment core team in moving forward to establish a useful set of questions.

As with every research opportunity, management is tasked with determining the primary drivers of change. This typical cascading of the analysis drives the next level of questioning. Assessment core team members then began to brainstorm what questions must be answered to resolve the puzzle. In order to accomplish this step, the maturity model was divided up among the assessment core team based on areas of interest and expertise (see figure A1). Each team member was asked to answer the question, "what does management need to know to solve the puzzle?" A final set of questions answering the question, "what do I need to know to answer the management question?" was derived as the assessment questions.

Each of the assessment questions were then rewritten as a positive statement. Use of the positive statements to which a respondent may agree or disagree allows for the data collection based on the Likert scale. The Likert Scale is normally considered to be ordinal data, or data that establishes a ranked order without a scaled value separating the rankings; however, the scale may be converted to numerical values, or interval level data (Romano, Kromrey, Coraggio, & Skowronek, 2006). The rankings of strongly-disagree, disagree, neither agree or disagree, agree, and strongly-agree are then coded as;

```
strongly-disagree = 1
disagree = 2
neither agree or disagree = 0
agree = 3
strongly-agree = 4
```

In order to use this process, the team must first accept the assumption that there is no relevant difference in the respondent's reasoning that would create a significant difference between strongly-agree and agree, or strongly-disagree and disagree (Romano, Kromrey, Coraggio, & Skowronek, 2006). This coding scheme allows for the data to be quickly assessed using quantitative statistical descriptive calculations of mean, mode, median, to determine the level of central tendency in the data. This process also allows for the use of demographic information such as project specific differences, employee role differences, and differences between management and employee opinion using simple statistical methods such as a t-test. The t-test checks to determine if there is a significant difference in the average difference-from-the-mean of one group from the difference-from-the-mean of another.

The assessment core team, using the focus area assignments from Figure A1, created over 400 positive statements. Statements were then evaluated and assigned to one of the maturity levels in the model. Using the maturity level assignments for each of the 400 positive statements in the evaluation, the maturity model was then completed. The assessment core team members each took the questions they created and presented the questions to 3 to 5 knowledgeable engineers in the department to validate that the questions make sense to potential respondents. Any questions raised during the test of the questions were addressed by adjusting the wording of the question. After the field test of the questionnaire a pilot test was conducted by asking each of the questions in a formal setting and comparing the questions for consistency across the pilot sample. Any additional questions were resolved by making small adjustments in the wording. This additional test also allowed the team members to practice the presentation, and prepare necessary clarifications.

Each statement was presented to a sample of 60 respondents out of the population of approximately 200 employees and contractors assigned to the CQA department. Responses for individual statements were averaged using an arithmetic mean, and for any statement averaging equal to or greater than 2.5, the statement was then tagged (green) as *achieved* at a department level. Any question averaging below 2.49 or below the question was tagged (red) as *not achieved*. Each statement was also evaluated to determine the percentage of respondents that answered the question of those present to answer the question. See Figure 2 for a typical question outcome. Once the department level analysis was complete, the data was further evaluated using the same process by employee role, group assignments, and priority projects to identify pockets of excellence or urgency.

1	SLA's & OLA's		Foundation	% responded
Foundation	1	My work partners are clearly defined for the project(s) I am working on. *NOTE: work partners include customers and suppliers.*	2.98	100%
Foundation	5	An escalation process is defined for the project(s) I am working on.	2.19	100%

Figure 2. A Typical Assessment Question Outcome

Note: Data in this figure is not from the actual assessment (mock data).

For any statement that is red, *not achieved*, the positive statement was then converted to a statement of opportunity. Each of the statements of opportunity were then grouped by subject as well as parameter (see Figure A1) and prioritized using a weighting scale based on difficulty-of-implementation, impact on the organization, and effect on cost, quality, and schedule (see Fig-

ure 3). Cost, quality, and schedule were each valued as 1 if the assessment core team members believed the recommendation will have an effect in that area, and valued as 0 if it was believed there was little value add. The impact column was valued at 1, 3, or 9 based on perceived value, and the difficulty was valued as 1, 3, or 5. Each of the values were evaluated subjectively as a perceived value by the assessment core team members.

The impact score was ranked as follows;

Low Impact on department ability to perform = 1

Medium Impact on department ability to perform = 3

High Impact on department ability to perform = 9

The Difficulty score was ranked as follows;

Difficult to Implement = 1

Medium difficulty to Implement = 3

Easy to Implement = 5

Score	Cost	Quality	Schedule	Recommendation	Impact	Difficulty
135	1	1	1	An escalation process should be defined for project roadblocks and risk mitigation.	9	5

Figure 3, Recommendation Prioritization Table

Note: Data in this figure is not from the actual assessment (mock data).

Prioritization Score is calculated as

P = (C + Q + S) x (I x D)

Where

P = Prioritization Score

C = Cost of company projects and product releases

Q = Quality of department project outcomes and company product releases

S = Department ability to maintain a project schedule

I = Impact on the overall department capability

D = Difficulty to implement

After the Assessment

Recommendations with top prioritization scores were combined into like recommendation groups. Those projects with the highest prioritization scores and with high impact and ease of implementation were scheduled to be accomplished in the first quarter of 2011. For each of the 1Q11 projects the assessment core team conducted a stakeholder analysis to identify those employees and managers with a direct stake in the outcome of the change. Teams were formed with each team assigned a facilitator, team lead, and champion.

Facilitators were assigned as an assessment core team member for each project with the role of ensuring the use of effective change management tools and practices, and maintaining a complementary relationship with the other team goals. Team leads were tasked with creating team work schedules, assignments, and reporting to the assessment core team, and CQA department management team on team progress. Champions were recruited for each team from the CQA senior management team to ensure teams were properly resourced, roadblocks do not exist, and department goals remained in focus throughout the project.

Teams then validated that a change may be created without breaking something, and a challenge statement was crafted to guide the team throughout the exercise. Each team's first work assignment was to create and report to the department leadership team a project charter to ensure clarity in project goals and full and shared understanding of the recommendations. Using this framework team members remained motivated and focused on the task creating effective changes that are well integrated into the department's goals while dedicating approximately 4 hours each week to the team.

Reports were provided to O'Leary as guides in the change implementation process to assist in planning for change and assessing the growth path for the department. Recommendations were categorized into areas regarding factors that may affect Cost of Quality, Productivity and Automation, Offshore Leveraging, Schedule and On-time Delivery, Estimation Accuracy, and Engineering Skills and presented as Spider Charts for further evaluation. See Figures 4 through Figure 10 to see the alignment of parameters with focus presentations.

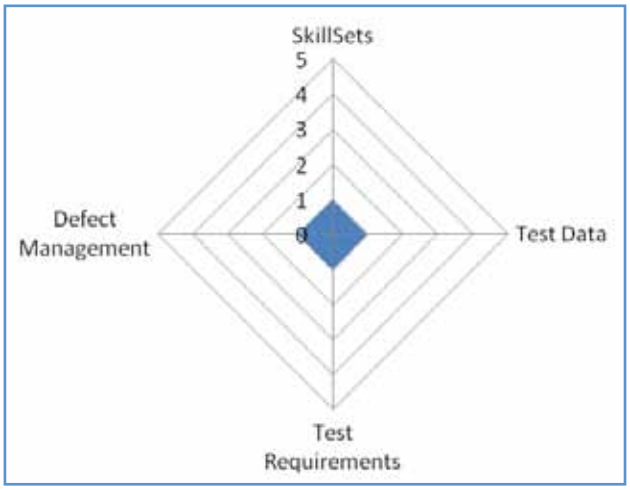


Figure 4, Areas of Focus that may Affect Cost of Quality *

Through the presentation of these charts, the management team is able to see how the sub-focus area parameters relate to specific activities and capabilities of the organization, and allows for prioritization of change.

Figure 5, Areas of Focus that may Affect Productivity and Automation *

Figure 6, Areas of Focus that may Affect Offshore Leverage *

Figure 7, Areas of Focus that may Affect Schedule and On-time Delivery *

As the management team reviews the charts, areas of strength become apparent, and the need for change may be readily identified regarding a specific parameter that may impact several high priority focus areas such as increased engineering skills.

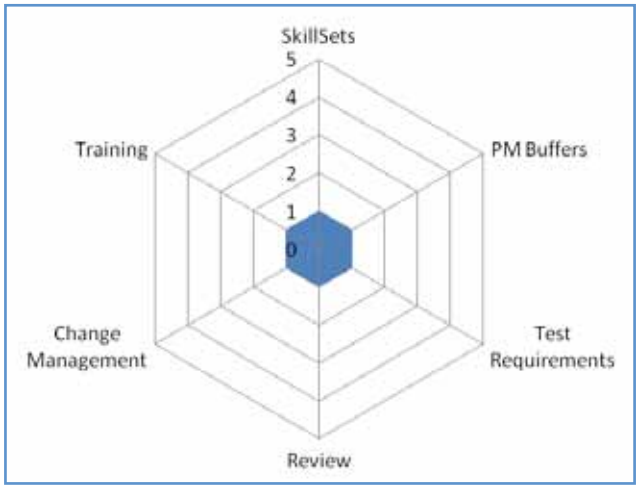


Figure 8, Areas of Focus that may Affect Estimation Accuracy *

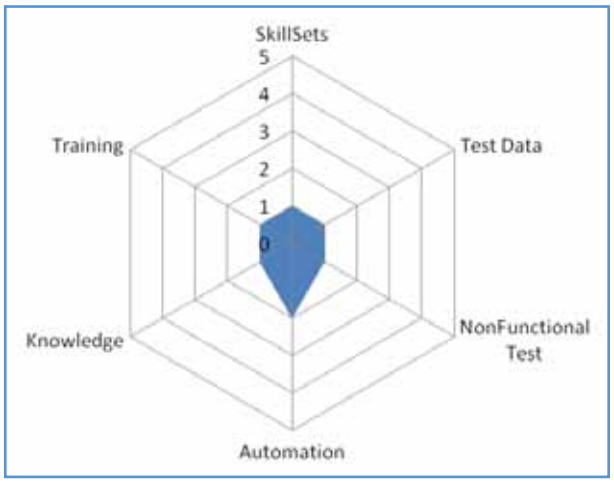


Figure 9, Areas of Focus that may Affect Engineering Skills *

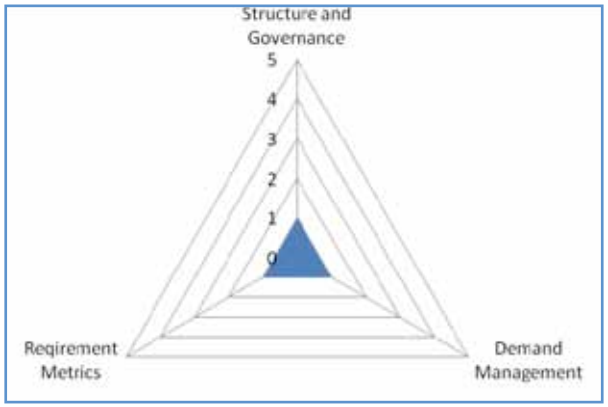


Figure 10, Areas of Focus that may Affect Non-Functional Testing *

The first quarter of 2011 was spent laying the baseline for the rest of the year. Teams successfully completed changes on the subjects of project communication plans, standard metrics and reporting, QA involvement in static test planning, standardized project planning methods and tracking, and risk based test planning. Through the team process, buy-in to the changes have become a peer pressure event. Team members have a lot of confidence in the team based change process and a personal stake in the adoption of their new tools and procedures. The changes, however, did not come without a few bruises.

Lessons Learned

Training is important.

Each of the teams experienced a lot of success, but many team hours were used in agreeing to the processes that would be used in identifying methods and resolving discussions regarding the usefulness of standard methodologies. When looking back at the first quarter, if time had permitted, we would have scheduled training in the proposed methods and tools. Starting in April 2011 we have begun a training program in 6 Sigma methods, and have a target goal of training 100% of our full-time employees.

Training may also lessen another common event in team based change. Often, when employees have a passion for making positive change, the common issue of attempting to fit a preferred solution to an ill-fitting opportunity can create bumps along the way to success. A couple of our teams did have to work through several pet solutions before finding the best solution and working to an effective change. This, again, may be alleviated through

a solid training program prior to starting the team building process.

The Remainder of 2011

We are now in the process of forming teams to tackle the proposed recommendations for 2Q11. Cognizant is now on board to provide us with the experience and professional coaching that is required for any company to accelerate any change program. Following our training program in 6 Sigma processes, and the addition of several more experienced change facilitators from Cognizant, we are anticipating a very successful and fruitful outcome to a very ambitious change program. A second assessment is planned for 3Q11 to assess the department direction and effectiveness of change in the first two quarters of the year.

References

Basili, V.R., Caldiera, G., & Rombach, H.D. (2002). Experience factory. Encyclopedia of Software Engineering. John Wiley & Sons, Inc. DOI: 10.1002/0471028959.sof110/full

Grunbacher, P., Halling, M., Biffel, S., Kitapci, H. & Boehm, B.W. (2004). Integrating collaborative processes and quality assurance techniques: Experiences from requirements negotiations. *Journal of Management Information Systems*. 20(4), 9-29. Retrieved from http://wv9lq5ld3p.search.serialssolutions.com.library.capella.edu/directLink?&atitle=Integrating%20Collaborative%20Processes%20and%20Quality%20Assurance%20Techniques%3A%20Experiences%20from%20Requirements%20Negotiation&author=Paul%20Grunbacher%3B%20Michael%20Halling%3B%20Stefan%20Biffl%2C%20Hasan%20Kitapci%3B%20Barry%20W%20Boehm&issn=07421222&title=Journal%20of%20Management%20Information%20Systems&volume=20&issue=4&date=20040401&spage=9&id=doi:&sid=ProQ_ss&genre=article&lang=en

Kruchten, P. (2000). Software development best practices. In G. Booch, I. Jacobsen, & J. Rumbaugh (Eds.), *The rational unified process an introduction* (2nd ed.) (pp. 3-16). Upper Saddle River, NJ: Addison-Wesley

Sterman, J., Kofman, F., & Repenning, N. (1997). Unanticipated side effects of successful quality programs: Exploring a paradox of organizational improvement. Management Science. 43. Retrieved from http://web.mit.edu/jsterman/www/ADI.pdf

Ward, J. & Peppard, J. (2002). *Strategic Planning for Information Systems* (3rd ed.). New York: John Wiley & Sons, Ltd. ; West Sussex, England.

* Note: Data is not representative of Comcast and is mock data for purpose of illustration.

Sub-Focus Area	Parameter	Parameter	Parameter	Parameter
Structure and Governance	SLAs & OLAs	Program Engagement	Communication	Organization Structure
Operating Model	Demand Management	Project Management		
Skill-Sets	Role-Skill Mapping	Competency Baselines	Skills Database	
Training	Training Strategy	Training Coverage		
Release Management	Release Planning	Release Readiness Tracking		
Configuration Management	Tool Definition	Access Levels	Variation Log	Code Promotion
Change Management	Coordinate Implementation	Coordinate Build & Test	Change Authorization	Change Definition/Log
Defect Management	Defect Prevention	Defect Containment	Cost of Quality	
Review	Review Lifecycle	Review Efficiency		
Knowledge Management	Assets Created	ReUsed	Contributing	Recommended Assets ReUsed
Risk Management	Risk Mgt. Process	Risk Identification	Risk Mitigation Efficiency	
Test Requirements	Requirement Metrics (RSI)	Requirement Classification		
Test Strategy and Planning	Testing Scope	Testing Approach	Testing Types and Methods	Entry and Exit Criteria
Test Estimation	Effort Overrun			

Sub-Focus Area	Parameter	Parameter	Parameter	Parameter
Test Design	Test Case Selection	Test Case Design Technique	Test Case Prioritization	Test Case Maintenance
Automation	Functional	BVT, Sanity	Regression	Process Automation
	Batch	Test Management	Data	
Nonfunctional Testing	NFT Requirements	Baselining and Benchmarking	Dedicated Environments	NFT Strategy
	Tools	Functional Segregation		
Test Data	Test Data Strategy	Tooling Strategy	Data Governance	ReUse
	Organization and RACI			
Environments	Environment Strategy	Sizing	Hardware & SW Provisioning	Support Process
	ReUse Strategy & Demand Mgt			

Figure A1, Assessment Matrix

> biography

Tom Wise
is Director of Quality Management at Comcast Corporation's National Engineering and Technical Operations National Lab, and was recently Sr. Manager Options Quality at Philadelphia Stock Exchange, and Associate Director Quality Management for the Chicago Mercantile Exchange. The author is a PMI Certified Project Management Professional, and ASQ Certified Manager of Quality and Organizational Excellence, and holds Bachelor's degrees in Organizational Management and Human Resource Management, an MBA in Management, and a Doctoral candidate in Organizational Management at Capella University. Most recently the author is teaching Software Testing at Villanova University in the Master Science program for the Computing Sciences Department.

Masthead

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

350.000 readers

ADVERTISEMENTS
sales@testingexperience.com

SUBSCRIBE
www.testingexperience.com/subscribe.php

PRICE
online version: free of charge -> www.testingexperience.com
print version: 8,00 € (plus shipping) -> www.testingexperience-shop.com

ISSN 1866-5705

Index of Advertisers

Berlin, Germany

IT Law
Contract Law

German
English
Spanish
French

www.kanzlei-hilterscheid.de
info@kanzlei-hilterscheid.de

k a n z l e i h i l t e r s c h e i d

Training with a View

Díaz Hilterscheid

20.06.11–22.06.11	Certified Tester Foundation Level - Kompaktkurs	German	Berlin
27.06.11–01.07.11	Certified Tester Advanced Level - TEST ANALYST	German	Berlin
28.06.11–30.06.11	Certified Professional for Requirements Engineering - Foundation Level	German	Mödling/Austria
04.07.11–08.07.11	Certified Tester Advanced Level - TESTMANAGER	German	Munich
05.07.11–08.07.11	Certified Tester Foundation Level (Summercamp)	German	Berlin
11.07.11–14.07.11	Certified Tester Foundation Level	German	Frankfurt am Main
11.07.11–15.07.11	Certified Tester Advanced Level - TESTMANAGER (Summercamp)	German	Berlin
13.07.11–15.07.11	Certified Professional for Requirements Engineering - Foundation Level	German	Berlin
18.07.11–22.07.11	Certified Tester Advanced Level - TEST ANALYST	German	Düsseldorf/Cologne
18.07.11–22.07.11	CAT - Certified Agile Tester (Summercamp)	English	Berlin
25.07.11–27.07.11	Certified Tester Foundation Level - Kompaktkurs	German	Berlin
01.08.11–05.08.11	Certified Tester Advanced Level - TECHNICAL TEST ANALYST	German	Berlin
01.08.11–04.08.11	Certified Tester Foundation Level	German	Munich
08.08.11–10.08.11	ISEB Intermediate Certificate in Software Testing	German	Berlin
11.08.11–11.08.11	Anforderungsmanagement	German	Berlin
15.08.11–17.08.11	Certified Tester Foundation Level - Kompaktkurs	German	Berlin
15.08.11–19.08.11	CAT - Certified Agile Tester	English	Berlin
15.08.11–19.08.11	CAT - Certified Agile Tester	English	Helsinki/Finland
22.08.11–26.08.11	Certified Tester Advanced Level - TESTMANAGER	German	Frankfurt am Main
29.08.11–01.09.11	Certified Tester Foundation Level	German	Düsseldorf/Cologne
31.08.11–01.09.11	HP Quality Center	German	Berlin
31.08.11–02.09.11	Certified Professional for Requirements Engineering - Foundation Level	German	Mödling/Austria
05.09.11–09.09.11	Certified Tester Advanced Level - TESTMANAGER	German	Berlin
05.09.11–08.09.11	Certified Tester Foundation Level	German	Mödling/Austria
12.09.11–15.09.11	Certified Tester Foundation Level	German	Munich
12.09.11–16.09.11	Certified Tester Advanced Level - TEST ANALYST	German	Berlin

- subject to modifications -

more dates and onsite training worldwide in German, English, Spanish, French at http://training.diazhilterscheid.com/

iseb Accredited Training Organisation

IREB International Requirements Engineering Board

CAT Certified Agile Tester®

ISTQB® Accredited Training Provider

Kurfürstendamm, Berlin © Kai-Uwe Schülke