

The **INTERNATIONAL** Monthly On Software Testing

Tea-time with Testers

Jerry Weinberg

Why Not Just Test Everything?

James Christie

Why Do People Happily Accept Poor Quality?

Dr. Meeta Prakash

What holds for you in the Cloud?

Markus Gärnter

Testing and Management Mistakes

New!
Testing Puzzles
by **Sebi**

Nathalie Rooseboom de Vries

The Joke's On You!

T Ashok

The Diagnosis

Teach-Testing →

An Ambitious Campaign by Tea-time with Testers

Petteri Lyytinen

Test Cases In Agile - A Waste Of Time ?

Joel Montvelisky

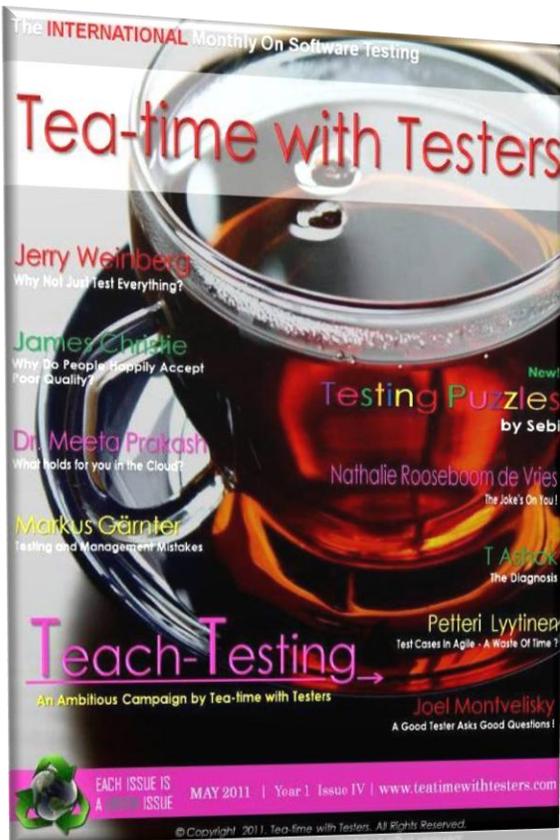
A Good Tester Asks Good Questions !



EACH ISSUE IS
A **GREEN** ISSUE

MAY 2011 | Year 1 Issue IV | www.teatimewithtesters.com

© Copyright 2011. Tea-time with Testers. All Rights Reserved.



Dear Readers,

A couple of months ago I had presented keynotes in 2nd Monthly Meet of Mumbai Testers. Topic of my speech was *educating the engineering students as well as students from other faculties with comprehensive knowledge of Software Testing*. The audience out there appreciated my thoughts and congratulated me for my sincerity behind contributing to the testing community via Tea-time with Testers. I felt delighted but that was definitely not the result for which I had presented my thoughts.

Whenever I look back into the past, the only thing that disturbs me most is the *fact of not getting to learn anything about software testing when I was an Engineering student*. I am pretty sure that I could have understood more things well before, than what I know after spending two years in the field of software testing.

Every second week I get to read about Fake Resumes, Fake Testers, Bad Interviews & stuff like that but my question is *why this situation even arises?* Don't you think that person will fake only when he knows that *he knows nothing?* Will anyone ever fake when he/she has enough knowledge and skills to prove the caliber? I personally feel that bad practices come into existence out of ignorance about the things. Let it be the ignorance of an interviewer or that of the one who gets interviewed, root cause of this ignorance is lack of education. Now, one would debate that we are not supposed to rely only on college education, we should study our own. But when things are possible quite before you enter in the industry, why not to?

The current situation is engineering students don't pass out with the mindset of choosing s/w testing as a career. Why will they, when they don't even know what s/w testing is all about? *If students are getting to learn s/w testing in a poor book of 100 pages, that too in last semester, why will they even think about taking it as a career?*

Well, what is the reason of sudden boom in Private Software Testing Teaching/Training Institutes in India? What different things they teach there which can't be taught to students in colleges?

Friends, our thinking time is over and this situation now needs some action. I proudly declare that *Tea-time with Testers* has already taken it. Yes, we are straight there in ground with our **Teach-Testing Campaign**. What all I request is just your **VOTE** and **VOICE**. *We are going to make a report of this campaign and send it to various Engineering Colleges as well as Universities*. If possible, to NASSCOM too.

Tea-time with Testers is determined to put efforts on it. Let's join hands and make this happen. There is big generation behind the door which deserves this justice! Enjoy Reading!

Yours Sincerely,

Lalitkumar Bhamare



Created and Published by

Tea-time with Testers.

Hiranandani, Powai- Mumbai -400076
Maharashtra, India.

Editorial and Advertising Enquiries:

Email: teatimewithtesters@gmail.com
Pratik: (+91) 9819013139
Lalit: (+91) 9960556841

© Copyright 2011. Tea-time with Testers.

This ezine is edited, designed and published by **Tea-time with Testers**. No part of this magazine may be reproduced, transmitted, distributed or copied without prior written permission of original authors of respective articles.

Opinions expressed in this ezine do not necessarily reflect those of editors of **Tea-time with Testers** ezine.

QuickLook



Testing
PUZZLES
by Sebi

Editorial

What's making News?

Tea & Testing with Jerry Weinberg

Speaking Tester's Mind

Why Do People Happily Accept Poor Quality?
-15

Testing and Management Mistakes: Causes
- 19

The Joke's On You -24

In the School of Testing

What Holds For You In The Cloud? -29

Test Cases in Agile -A Waste of Time?-32

Testing Intelligence: A Good Tester Asks
Good Questions - 36

The SCRUM Primer (Part 4) 39

T' Talks

The Diagnosis - 46

Tool Watch

Testing Puzzle - S.T.O.M. Contest

Our Testimonials

Family de Tea-time with Testers

What's making News?

- find out the latest happenings in the technology world

Have you got a BUG?



If you think software testing is layman's job, think twice! Software Testers are now considered as a commodity. There is tremendous demand for software testers and job opportunities in Software testing across the globe are on rise, finds Alap Patel.

A submission by *Tea-time with Testers* Fan.

Time is changing and so is the Industry Outlook for the field of Software Testing. It used to be defined as a job of finding bugs in software. But wait! Software Testing is not just about finding bugs in your software under test but to provide stakeholders with information about the quality of the product and its functioning too.

CHANGING ERA OF SOFTWARE TESTING

Cloud-based applications are becoming a trend in the software market. As a result, the testing market is also cloud-based. Companies such as IBM, CSS Corp, TCS etc. have started cloud-based testing, which is likely to reduce the cost and timing of testing the application. Companies are significantly in building competence in various specialized testing services such as enterprise resource planning (ERP) testing i.e. Oracle Siebel application testing, web-based application testing, service oriented

architectures (SOA) testing, software-as-a-service (SAAS) testing, wireless and mobile application testing etc. "As the domain of software application development is increasing significantly, so is the need for testing specialized applications," says the report. ([Reference](#))

TESTERS AS A COMMODITY

After interacting with my colleague he said that, "Companies in India look for quality instead of quantity that is why the other countries get attracted to outsource major amount of work to the Indian IT sector". A friend of mine who has worked in Finance Sector earlier and now perusing his MBA says that, "Companies are on cost cutting globally and they do not want to spend more on software testing. Indian Consultancy companies offer these services at low cost by recruiting bulk of Testers." One Mumbai based career counselor had said that, "With more complicated programming languages and different coding schemes, software testing has become a critical job for the companies. Almost every large IT Company has more than 18 percent of the total workforce dedicated for the software testing; a lot of small companies also come up with the only Software testing solutions."

BOOMING JOB OPPORTUNITIES

Apart from the IT companies other Industries also require the skilled software testers, like investment banks, various consultancy firms, retailers, airline Industry and networking companies. Now the Testing is seen as the high profile corporate job, which has well defined career path e.g. a test engineer eventually can become a Test Analyst, Senior Software Engineer and Test Manager. If you know the functionality of what you are testing in better way then you can have better growth.

BASIC REQUIREMENTS

A Btech/BE in Computer Science/ Information Technology or any stream, or MCA can be applicable for the any testing jobs. There is different area in Software testing like Manual Testing, Automation Testing and Performance Testing etc. Companies mainly recruit for the manual testing for which it requires good functional knowledge of the domain, analytical abilities, and good problem solving capabilities with good communication skills. Companies who recruit the testers for the Automation testing should have the Experience, good knowledge of programming languages like C, C++, C#, JAVA, Scripts and knowledge of various Testing Tools.

SKILLS THAT YOU NEED TO HAVE

Special testing skills are in demand now. Generally there is lack of skills in the various areas of test automation and scripting. Also demands are on rise for technical skill sets like the ability to review and manage various automation tools, generate test automation frameworks and figure out how to leverage reusable on-demand components. In a same way there is always consistent demand for the performance tester to enhance the performance of the application, to offer suggestions. People having good command over HP QuickTest Professional (QTP), Rational Robot, Microsoft Application Center Test (ACT), HP WinRunner and also tools like IBM Rational Functional Tester and HP LoadRunner always are in great demand.

CERTIFICATIONS

Various certification programmes are there to test the software testing skills of a tester.

Following are the list of testing certificates.

- The International Software Testing Qualifications Board (ISTQB)
- Certified Associate in Software Testing (CAST) offered by the Quality Assurance Institute (QAI)
- CAT offered by the International Institute for Software Testing
- Certified Manager in Software Testing (CMST) offered by the Quality Assurance Institute (QAI)
- Certified Software Tester (CSTE) offered by the Quality Assurance Institute (QAI)
- The Certified Software Test Professional (CSTP) offered by the International Institute for Software Testing and many more.



- Alap Patel, Mumbai-India



Now there is **Testing Puzzle** in **Bug-Boss** Challenge! Are you Ready?

Scroll down to our **Testing Puzzle** Page and claim your **Smart Tester Of The Month** Award!

Do you think that Software Testing should be taught comprehensively in engineering colleges as well as a separate course under universities?

then extend your hand...

join us in...

Teach-Testing

An Ambitious Campaign by Tea-time with Testers

Cast Your Vote & Raise Your Voice !

It's definitely going to make the difference!

[Know More](#)

 [Click here](#) To Cast Your Vote and Send Your Ideas!

Subscribe [here](#) Right Away to get our all Issues for FREE

www.teatimewithtesters.com

Tea & Testing



with

Jerry Weinberg

Why Not Just Test Everything?

I was called in to consult recently by a test manager whose manager had demanded that he "test everything." It was not the first time I'd heard this impossible demand.

Why is it impossible? First of all, the human brain not only makes mistakes, its capacity is finite. Second, nobody lives forever. So, as much as we would like to perform all possible tests, we can't think of them, and, even if we could, we wouldn't live long enough to do them all. Besides, for most situations, it would cost too much—since the number of possible tests for any given program is infinite. Let's see why.

There are an infinite number of possible tests.

Let's think of the simplest program we could conceive of to test: a program whose function will be to respond to tapping on the space bar by putting a "Hello Dolly!" message on the screen. What would we test for? To keep it simple, we'd want to test that every time we pressed the space bar, we got the "Hello Dolly!" message, and every time we pressed any other key or combination of keys, we never got anything on the screen.

If you can't see inside the program, how many test cases would you need to execute in order to bet your life that you've tested all possibilities? Identify the number of test cases, write that number down, and then read the next paragraphs to see if you got it right.



Since you can't see inside the program, you have no idea what bizarre conditions the programmer might have set up. For example, the program might be designed so that everything looks normal on the outside unless the user types an extremely unlikely sequence of keystrokes. Suppose I set up the program so that if you hit, say, the W key, then the space bar three times, then the M key, then the space bar another three times, then the J key, then you type exactly 168 more keystrokes without once using the letter L, the program will put a message on the screen asking, "Whaaa?" Would your exhaustive set of test cases have detected this outlandish condition in which an unwanted and unexpected Whaaa response is hidden in the program?

Do you think these conditions are outlandish? Unrealistic? During a technical review of a supposedly highly secure application, we discovered that a programmer named Wanda Marilyn Jones (not her real name, but her real initials) had placed exactly this backdoor into the software, writing the password protection so that she could bypass the ordinary password protection, regardless of what the real password was set to be, thereby enabling her to break in at any time. A highly sophisticated test plan, executed under strict controls, had not found this backdoor, which lay hidden for three years until we performed a technical review. To paraphrase Edsger Dijkstra, "Testing can reveal the presence of bugs, not their absence."

Do you get the point by now? If you didn't guess that the number of tests required to exhaustively test software is infinite, or at least "a number greater than I could run in my lifetime," you didn't understand the point of this article. Now you do.

Notice that I didn't even mention the possibility of testing this program on different configurations, a problem commonly faced by product developers. If a program has to work on ten different CPUs, each with ten possible memory sizes and ten different disk-drive sizes, that would mean 10 times 10 times 10 different configurations that would have to be tested.

But that's way too simple for many real testing situations, where the tester has to deal with the added complexity inherent in having different manufacturers, drivers, operating system versions, other programs simultaneously operating, and combinations of other different peripheral devices, any of which could contain errors. Dealing "completely" with all possible configurations like this would necessitate quadrillions of different test cases. And that's not considering tests for all the different functions that program is supposed to perform with any of these configurations.

Even this immense number ignores sequence effects reflective of the order in which the tests are performed. If there are ten functions a user might invoke, it isn't enough to use ten different tests because they might produce different results if performed in different orders. So, instead of ten tests, we'd need ten factorial (10!) tests (over six million) tests to cover all sequences.

But that wouldn't be enough, either, because if the program has memory (and all real programs do), then the second time we perform a test sequence, it may not produce the same results as it did the first time. And these immense numbers (all multiplied together) also ignore true randomness, like the exact nanosecond an external device causes an interrupt or the exact timing to the microsecond of when you strike, say, the J key. All in all, testing can be exhausting, but it can never be exhaustive.

Testing is, at best, sampling.

Since we can't test everything, any set of real tests is some kind of sample—a portion, piece, or segment that is in some way representative of a whole set of possible tests. We, of course, hope it's a good representative, but that brings up the question, "Good for whom?" Fundamentally, sampling is yet another psychological process—and an emotional one. A sample that satisfies one person may not satisfy another in the slightest.

So how do we decide what to sample? How do we know we're taking a large enough sample to adequately represent everything? How do we know we've taken an appropriate sample?

I was musing about this problem with my colleague Elisabeth Hendrickson as we watched my rain gauge during a rainstorm in Pecos Canyon, New Mexico. The gauge, which is attached to the exterior of my porch, had a small opening through which it was supposed to sample rain—which, at the time, consisted of large, widely spaced drops splattering on the ground every few seconds. It would have been an adequate rain gauge for Seattle, because it would have done well with a fine, misty rain that fell for hours, but when the Pecos storm stopped after ten minutes, the bottom of my gauge was completely dry, belying the fact that any rain had fallen.



But Elisabeth and I had seen the rain fall and, in fact, we were soaked. We quickly realized that the gauge was taking an inadequate sample when dealing with such huge drops falling several inches apart over the minutes the storm lasted. Elisabeth looked at my dripping beard and said, "You are a better rain gauge than the one on the porch."

Notice that it could have worked the other way. If just one or two of those huge drops had happened to fall into its small opening, the gauge might have reported a full quarter-inch of rain, which wasn't accurate either. We regularly see this same phenomenon in testing: We take a small sample—try a few things here and there—and end up under- or over-reporting the density of problems in the whole product.

The cost of information can exceed the cost of ignorance.

The impossibility of exhaustive testing squeezes us between two difficult and simultaneously desirable objectives:

1. We want to cover all interesting conditions.
2. We want to reduce the set of tests to a manageable, affordable level.

To understand what I mean by the first, consider the number of times testers stumble across a critical bug when they aren't looking for that particular type of bug. They find it because they are lucky (or unlucky, if they don't want to know about it), not because they were executing a set of meticulously designed tests intended to find that specific problem. The bug just appears, like an ant in your raisin bran. But is it pure luck? Is there some psychology to figuring out how to find more of these surprise bugs? I believe that part of the answer lies in expanding our idea of testing.

We can obtain more information with less testing— perhaps.

These days, many of the people I talk with are concerned with the second objective stated above—reducing the set of tests to a manageable, affordable level. They're asked, or commanded, to subsist and even thrive with smaller teams and greater responsibilities.

In one of the more extreme cases described to me, a tester sought a consultant's advice on handling the following dilemma: "We were just downsized from a team of thirty testers to three, but we're still supposed to 'ensure' the product. How do I decide what to test?"

One argument would say that testers can't "ensure" anything at all so they shouldn't even try. But that argument won't persuade an executive staff struggling to keep a firm afloat during rough economic times. So what to do? Admittedly, a downsized team can't do everything the larger staff used to do, but it can pick and choose from among the tests it could possibly perform. It can identify the tests that make the best use of limited resources.

The consultant's advice was grounding: "First of all, recognize that any set of tests is a sampling strategy and then, no matter how many or how few your resources, choose the best representative set of tests you can."

Imagine you are about to dine at the Testing Buffet.

You stand at the head of a long table full of use cases, boundary conditions, compatibility tests, interaction tests, permissions matrices, and so on, holding a single plate. The Testing Buffet allows a diner only a couple of trips through the line, so you know that you'd better choose wisely. What should you do?

Well, if you have ever watched people faced with this situation in a food buffet, you know that different personalities attack problems in different ways. Some people will complain to the maitre d' or waiter about the size of the plates and continue whining throughout the meal, spoiling everybody else's meal. Others will simply turn around and walk away in a huff because they believe they shouldn't be limited in the amount they may eat.

Some people will start at the head of the line and fill the plate with the first two dishes that appeal to them. Is this wise? Maybe at a restaurant, but probably not when testing with limited resources.

When faced with an insurmountable set of testing tasks (which is, really, always the case in testing), you may be tempted to begin at the beginning and see how far the testing progresses in the allotted time. Alternatively, you might pick and choose easy, quick tests across the entire feature set. Both approaches are convenient for the tester, but do they provide an adequate meal of testing?

To test well, testers must be aware of the constraints of finite tests, resources, and time. Testers must also be aware of their own personalities—the way they tend to attack the buffet.

Managers also must be aware of these constraints and tendencies. No matter how much you'd love the luxury, you can't expect testers to perform "exhaustive" tests. You'll have to reconcile yourself to satisfying your appetite for control in some other way.



Summary

There are an essentially infinite number of tests that can be performed on a particular product candidate. Rather than asking for "all" tests to be performed, managers and testers must strive to understand the risks added to the testing process by sampling.

Common Mistakes

1. Demanding "test everything": When you demand the impossible, you have no idea what you'll get—except that it won't be anything impossible.

2. Not understanding sampling: Very few managers (very few people, in fact) understand sampling very well. Either educate yourself or hire an expert to audit your sampling. In either case, always be ready for the possibility of a sampling error.

3. Spending too much for information that's not worth it: Do you have a basement or garage full of expensive gadgets that you never really wanted? Do you realize what else you could have done with the money spent (or the space occupied)? If so, you understand this error. Be careful what you ask for.



4. Testing for the sake of appearance: Some customers and certifying agencies demand "testing." You can go through the motions if you feel you must, but at least don't deceive yourself about the quality of the information you receive.

5. Not using all sources of information: Information gathered from test results is, by its very nature, limited, but there are other kinds of information sitting around if you're alert enough to see it.

6. Thinking that machines can perform exhaustive testing, even if people can't: It's not just the human brain that's limited; testing tools are limited, too. Don't buy any product that claims it can "perform all tests." Even if it could, you couldn't possibly look at all the results.

7. Increasing risk by constraining resources: When testing resources are cut, the easiest way to respond is by limiting sample size—running fewer tests. But with a reduced sample size, sampling errors become more likely. A diverse sample might find more problems than a large sample. Likewise, diversifying your test team might find more problems than enlarging your test team.

There is an exclusive review of our *April 2011* Issue by Mr. Jerry Weinberg and we have got Feedback from our readers too.

Do not forget to read our *Feedback and Responses* columns!

- Editor

Biography

Gerald Marvin (Jerry) Weinberg is an American computer scientist, author and teacher of the psychology and anthropology of computer software development.



For more than 50 years, he has worked on transforming software organizations. He is author or co-author of many articles and books, including *The Psychology of Computer Programming*. His books cover all phases of the software life-cycle. They include *Exploring Requirements*, *Rethinking Systems Analysis and Design*, *The Handbook of Walkthroughs*, *Design*.

In 1993 he was the Winner of The **J.-D. Warnier Prize for Excellence** in Information Sciences, the 2000 Winner of **The Stevens Award** for Contributions to Software Engineering, and the 2010 **Software Test Professionals first annual Luminary Award**.

To know more about Gerald and his work, please visit his Official Website [here](#) .

Gerald can be reached at hardpretzel@earthlink.net or on twitter @JerryWeinberg

Perfect Software and other Illusions about Testing is Jerry's one of the best book.

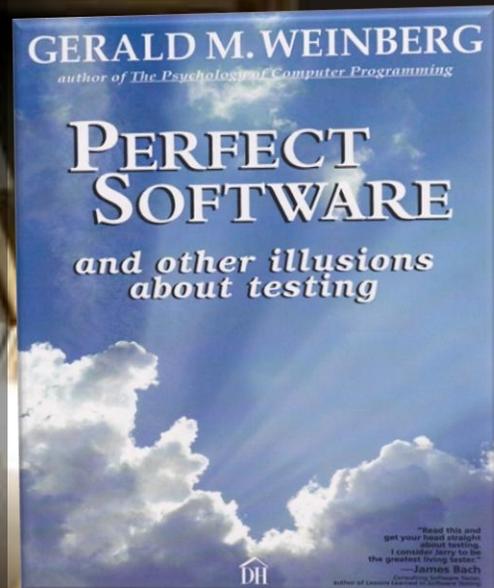
This book focuses on real time issues that testers face. Not only testers but entire project team should read this book.

Perfect Software changes the reader's way of looking at things. It answers all the questions that testers usually come up with.

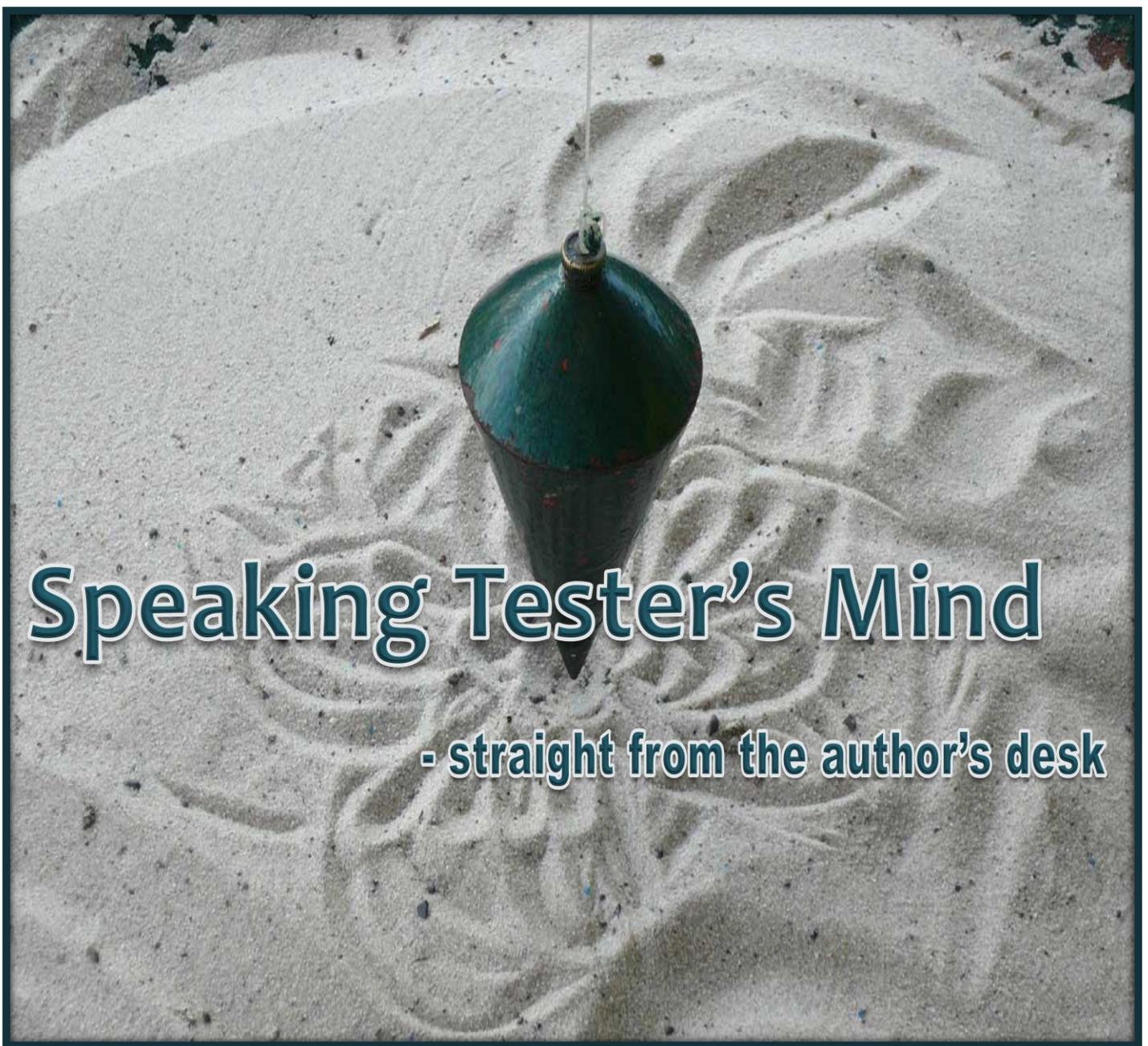
Tea-time with Testers recommends this book if you want to learn about bringing intelligence in your testing as well as to enhance your decision making ability.

Its sample can be read online [here](#).

To know more about Jerry's writing on software please click [here](#) .



TTWT Rating: ★★★★★

A photograph of a green, conical weight hanging from a thin string. The weight is positioned over a surface of light-colored sand. The sand has several circular, ripple-like patterns drawn into it, resembling a target or a series of concentric circles. The weight is centered over the middle of these patterns. The entire scene is framed by a dark blue border.

Speaking Tester's Mind

- straight from the author's desk



Why Do People Happily Accept Poor Quality ?

by James Christie

At the weekend whilst lounging in a hot bath, after an excellent dinner, I came across this striking article about kakonomics by Oliver Burkeman in the Guardian's magazine.

It sought to explain why so much of life just troubles. Given the extremely comfortable circumstances in which I was reading his article I wasn't entirely convinced that life does trouble all that badly. However, I certainly was receptive to the deeper meaning; that we are often motivated by laziness and the desire for an easy, unchallenging life.

People enter into a sly conspiracy in which they happily trade low quality results, whilst indulging in high quality, but bogus rhetoric, for the record.

Burkeman's article was based on the work of the Italian philosopher Gloria Origgi. See these articles from her blog; "Kakonomics, or the strange preference for Low-quality outcomes" and The Kakonomics of Facebook.

I had no difficulty accepting the gulf between grubby reality and the bogus rhetoric of high flying ideals. That has always been a feature of my working life.

Were people really happy with that gulf? It is of course counter-intuitive. I was sceptical. However, the more I thought about it the more willing I was to accept Origgi's insights.

Transformation Programmes that transform nothing

I have often been puzzled at how some organisations have been seemingly trapped in damaging, dysfunctional behaviour, with no apparent willingness to break out and transform themselves. They have certainly been too willing to launch Transformation Programmes (note the deliberately disrespectful and cynical capital letters).

These programmes have rarely been designed to truly transform anything. They have simply rearranged anything and everything capable of being rearranged. The real work has carried on in spite of the transformation.

Once I really did see one of those programmes produce genuine and exciting benefits. That those benefits were irrelevant and accidental by-products of the Programme was revealed shortly before it was wound down.

A review of what had been achieved swept the benefits away in an act of casual destruction that effectively reinstated the pre-transformation status quo, but with a superficially different structure and youthful new faces added to the lower layers of the management team.

Was kakonomics at work there? Did the consultants deliver some low grade, but expensive, rubbish, which the management were happy to take on board because it didn't challenge them to change either their behaviour or the organisation's culture?

The last minute review ensured that management could carry on as before, but with the enhanced prestige of having "done something" about the abysmal quality and late delivery of IT applications.

Kakonomics and traditional development practices?

I have worked on many developments when you could sense the relief at progress meetings when someone confessed that they'd had problems and would deliver late. By doing so they let the others off the hook and no-one would be required to deliver on time.

That is consistent with kakonomics, but it can also be attributed to a natural human desire not to be the first to admit failure, or an equally natural desire for easier targets. By their actions do people really solicit and encourage poor quality and performance?

The possibility that Origg might be right seemed stronger when I reflected on occasions when people had stood up and not only insisted on high quality outcomes, but delivered them. Were they lauded and rewarded as heroes?

Well, no. Not consistently.

I remember a friend once saying, shortly before he handed in his notice, that it made no sense to be a high performer at the company where we worked.

The difference in reward was negligible and insufficient to justify the extra effort. Worse, slovenly and inefficient work had its own reward in the form of generous, overtime without scrutiny of what was achieved. Sloppy cowboys could earn more than diligent craftsmen.

The users were untroubled. Everything was late and poor quality. However, the IT people made perfect scapegoats. Requirements could therefore be late, badly defined, and always open to change. IT would take the blame without complaint, largely because they knew it was expected of them, and in the full knowledge that the gap between aspiration and reality would be tolerated and leave them with a substantial comfort zone.

Both there and elsewhere rewards have gone more consistently to those who play the game rather than those who seek and even deliver genuine improvement.

Following practices, such as the Waterfall, V Model and Structured Methods, which are inefficient and ineffective but were widely regarded as "standard" or "best practice" for years reinforced kakonomic exchanges of low quality throughout the IT profession.

If people follow them and the results are bad then they can always pretend that they tried to do it right, but were unlucky. No-one is really to blame, or accountable, and we can all carry on as before. The rhetoric is impeccably high quality. The reality is garbage.

If people deliver something special and high quality then they are probably iconoclasts or rebels who have rejected the methods that would have ensured low quality failure. They have therefore rejected the cosy consensus of the status quo and set a challenge to the organisation.

Whereas those who fail do so conventionally, and can therefore be excused as unlucky but dependable, the rebels are seen as difficult, spiky individuals. Their success is a one off, which can't offer a pattern, being dependent on individual skill rather than repeatable process.

There's a blog to be written about whether CMMI is misused so that it encourages kakonomic exchanges by implicitly valuing repeatable, predictable failure above erratic, but sometimes brilliant individualism.

Anyway, the rebels have breached the subliminal contract of the kakonomic exchange, leaving their colleagues distinctly uncomfortable. No-one would admit to the real reason for that discomfort, and they happily ascribe it to concern about the rebels' cavalier style, which creates friction and harms morale. They are "difficult", "not team players".

The conventional team players glide up the organisation, and if the rebels are wise they head for an organisation where sparks are welcomed, rather than feared.

Is Agile a panacea?

I'm not sure there's much for proponents of Agile to be complacent about here. If Agile is badly implemented then I'd expect exactly the same reinforcement of bad practice. The trendier Agile becomes, the greater is the danger that it will be adopted for the bogus high quality rhetoric.

Perhaps Agile's greatest strength in the long run is its adaptability and flexibility, rather than any particular technique. Any shrewd practitioner should surely realise that complacency is deadly to Agile, and would be attuned to the dangers of kakonomic exchanges, even if the term and the theory meant nothing to them. In principle, I'd have thought that such exchanges could occur anywhere.

What about testers?

I think that kakonomics can explain some of the daft, damaging and dysfunctional behaviour that happens in organisations. It might explain why people persist in doing things that are entirely inconsistent with their supposed objectives, and why they will do so in the sure knowledge that their behaviour will not be challenged.

Biography



James Christie is a Chartered IT Professional, and professional member of the British Computer Society. He has 24 years commercial IT experience, covering test management, information security management, project management, IT audit, systems analysis and programming.

James is particularly interested in how the quality of applications can be improved by incorporating usability engineering and testing techniques. He is a member of the Usability Professionals Association.

He is enthusiastic about testing approaches that move beyond the traditional V Model and heavily scripted testing. He writes his thoughts on testing at his own blog <http://clarotesting.wordpress.com>.

James can be reached at james@clarotesting.com or on twitter [@james_christie](https://twitter.com/james_christie).

Gloria Origgi is rightly critical of the damage this all does. Her analysis is not just an amusing reflection on the vagaries of life.

She explains that the reason *"it is a form of collective insanity so difficult to eradicate, is that each low-quality exchange is a local equilibrium in which both parties are satisfied, but each of these exchanges erodes the overall system in the long run"*.

Testing should be a comfortable home for those who are happy with creating and suffering a little discomfort! It should be our job not to tell people what they want to hear, or to go through the motions, or to follow the process blindly, but to shine a light on what's really there.

In doing so it helps to think about why people behave as they do, and to understand why good people do bad things.

I don't think it always applies, and it's not a magical insight that explains everything, but maybe kakonomics is one of those things that can help us understand a little better, and to explain just why we are seeing damaging behaviour. It might not make us popular, but unless the organisation is dying there should always be a place for unpopular, truthful insights about why we get poor quality.

One of the things about testing that I like is that it is a profession that has genuine respect for the teller of those unpopular truths.

Do you have any Questions or Feedback on any article that we publish in Tea-time with Testers?

No Problemo ! We will be publishing your Feedback/Comments and even Answers to the Questions that you have for the Author on his/her article that gets published in our magazine.

Do write us with your Feedback and Questions (if any) in bellow format & send it to teatimewithtesters@gmail.com :

- Your Name
- Brief Introduction about Yourself
- Article Name
- Your Feedback/Questions

Make sure to write **Feedback for <Article Name>** in your Subject Line.

Testing & Management Mistakes : Causes

By Markus Gärtner

Michael Bolton wrote [a nice story](#) about a situation at work where the project manager asks a tester to come in for the weekend. I remember facing such a situation with a colleague at work. He was asked to stay in, and I explained to him some of the mistakes I saw. I told him about placating behaviour, and showed him blaming behaviour on the other side of the medallion. Since blaming probably leads to a showdown, I taught him how to improve the conversation using congruent communication. That is taking the self, the other, and the context position into consideration. If you are interested in more about this conversation model that Virginia Satir created, I whole-heartedly suggest you to read through Weinberg's [Quality Software Management series](#) ; especially **Volume 3** should be of interest.

I would now like to take a closer look on the underlying problems that make a tester react to management mistakes in an inappropriate way. By going over the dialogue of our manager, Magnus and our tester, Tim, we'll try to seek out for opportunities on how to solve the process problem.

Magnus the Project Manager: "Hey, Tim. Listen... I'm sorry to give you only two days' notice, but we'll need you to come in on Saturday again this week."

Tim the Tester: "Really? Again?"

Magnus plans in two days of lead time for Tim to re-arrange with his family in order to get to work on the weekend. This is surely a cause of Tim's behaviour in the previous weeks, as it turns out later. Magnus already became used to convincing Tim on short-notice to turn in for the weekend. Magnus became conditioned by Tim's earlier behaviour, just as [Pavlov's dog](#). **Operant Conditioning** might be a cure for this underlying cause here. If Tim would now force a punishment, the negative feedback loop may be broken up. Of course, over time Tim will feel enough pressure to bring in some form of negative reinforcement for Magnus – one way or the other. So, the root-cause here is placating behaviour from Tim as a reaction to the (almost) blaming behaviour from Magnus.



Magnus: *"Yes. The programmers upstairs sent me an email just now. They said that at the end of the day tomorrow, they're going to give us another build to replace the one they gave us on Tuesday. They say they've fixed another six showstoppers, eight priority one bugs, and five severity twos since then, and they say that there'll be another seven fixes by tomorrow. That's pretty encouraging—27 fixes in three days. That's nine per day, you know. They haven't had that kind of fix rate for weeks now. All three of them must have been working pretty hard."*

Tim: *"They must have. Have they done any testing on those fixes themselves?"*

Magnus is diving into interpretation here almost directly. The only thing that I can derive from the developers' thinking they have fixed that many bugs is that they worked on those bugs. It does not mean they worked pretty hard on it. It just means that the developers think they were fixing many serious bugs. Actually, "it's nothing until it's reviewed" goes also for bug fixes. It's nothing until it's tested. That said, Magnus dives into interpretation and judging mode far too soon. In addition he gets attracted by the numbers and the outlook of the promising delivery. Having faced some assessment and education on Myers-Briggs or another personality model, would reveal his self-blindness in this situation. Maybe Magnus can become more aware of his deficits by getting taught about his personality and acting accordingly.

Tim is just a tester. He may not experience consciously this flaw. But he has to suffer under the outcome of it. So, making Tim aware of the model and talking consciously with Magnus about the situation, his addiction to numbers in this particular circumstance, could help overcome his self-blindness. Though, Tim will need some advice on how to give that advice in a manner that Magnus will accept. Thus far, I never achieved this.

Magnus: *"Of course not. Well, at least, I don't know. The build process is really unstable. It's crashing all the time. Between that and all the bugs they've had to fix, I don't imagine they have time for testing. Besides, that's what we pay you for. You're quality assurance, aren't you? It's your responsibility to make sure that they deliver a quality product."*

Tim: *"Well, I can test the product, but I don't know how to assure the quality of their code."*

Magnus: *"Of course you do. You're the expert on this stuff, aren't you?"*

There are two messages I can derive from Magnus here, pointing at a serious, but far too common problem in software management. First, he sends out the signal to his developers that it is more important to fix bugs than to test them. By measuring the poor bug fixing rate, the emphasis for the project is clearly given in this situation. Second, he frees precious developer time from testing the fixes. Actually, he tells his developers, that it's more important to fix the bugs, rather than test them. This is a very bad combination, but far too common.

Tim reacts greatly here, but without Magnus listening to his point the effort is basically wasted. Magnus should take the time to listen carefully to his tester in this situation. This would not only raise the trust level for him in the eyes of Tim, but also make him aware of the most serious problems he's introducing. As a manager, you have to pay attention to what is said to you.

Tim: *"Maybe we could arrange to have some of the testing group go upstairs to work more closely with the programmers. You know, set up test environments, generate data, set up some automated scripts—smoke tests to check that the installation..."*

Magnus: *"We can't do that. You have high-level testing to do, and they have to get their fixes done. I don't want you to bother them; it's better to leave them alone. You can test the new build down here on Saturday."*

Tim seems to be very aware of Agile software development, and knows that co-locating the whole team is the right way to tackle the team problems they're facing caused by bad management. Again, Magnus is not listening to the points from his tester. Instead he gives in to react upon Tim's try to manage the project by asking for a change in the team settings. Far too often I have seen such reaction where the project manager explains what type of testing is needed. If the manager starts to make these decisions, you're seriously in trouble. A good testing lead would have explained Magnus the flaw of his assumption, that high-level testing is sufficient. I wonder when this message will reach management world-wide, as I see it far too widespread. "Perfect Software... and other illusions about testing" from Gerald M. Weinberg explains the underlying problems to this thinking.



Tim: (pauses) *"I'm not sure I'm available on Sa..."*

Magnus: *"Why not? Listen, with only two weeks to go, the entire project depends on you getting the testing finished. You know as well as I do that every code drop we've got from them so far has had lots of problems. I mean, you're the one who found them, aren't you? So we're going to need a full regression suite done on every build from now until the 13th. That's only two weeks. There's no time to waste. And we don't want a high defect escape ratio like we had on the last project, so I want you to make sure that you run all the test cases and make sure that each one is passing before we ship."*

Again, Magnus should pay attention to the project here. The cause here is the ignorance of relevant facts. With the help of systems thinking, he could see the outcomes of his management decisions here. Tim is not in charge to make the test cases pass. The bugs need to be fixed accordingly by the developers. But based on the previous statements, they got higher priority for fixing the remaining bugs. Weinberg called this reinforcing feedback loop that brings relief in short-term, but pain in the long-run an addiction cycle. Tim is doomed when the developers won't start to take care for the quality of their own product. The short-term relief caused by the mandated overtime for Tim does not pay this one off, so in the long-term the team will continue to struggle.

Tim: *"Actually, that's something I've been meaning to bring up. I've been a little concerned that the test cases aren't covering some important things that might represent risk to the project."*

Magnus: *"That might be true, but like I said, we don't have time. We're already way over the time we estimated for the test phase. If we stop now to write a bunch of new test scripts, we'll be even more behind schedule. We're just going to have to go with the ones we've got."*

"We don't have time to do X" almost always boils down to a management problem. Notice that Magnus uses this phrase quite often in the overall conversation. Bringing in some NLP-approach from Tim's side here could help. Magnus is probably put under pressure, either by time or by his upper management. But he should relax and take conscious decisions rather than giving in. The first thing that shuts down in a crisis is the measurement system as I learned from Weinberg in Quality Software Management Vol. 4. And the situation never improves if you shut your eyes when faced with problems. Looking away from the flaws in the written test scripts, will not improve them. Neither will it improve the quality of the delivery. If there is a problem with the time, Magnus should tell Tim to take the time to improve the scripts. Magnus appears to have not learned to resist this pressure.

Tim: *"I was thinking that maybe we should set aside a few sessions where we didn't follow the scripts to the letter, so we can look for unexpected problems."*

Magnus: *"Are you kidding? Without scripts, how are we going to maintain requirements traceability? Plus, we decided at the beginning of the project that the test cases we've got would be our acceptance test suite, and if we add new ones now, the programmers will just get upset. I've told them to do that Agile stuff, and that means they should be self-organizing. It would be unfair to them if we sprang new test cases on them, and if we find new problems, they won't have time to fix them. (pause) You're on about that exploratory stuff again, aren't you? Well, that's a luxury that we can't afford right now."*



From this statement it's obvious to me, that Magnus just jumped on the Agile wagon without understanding, what it means. One of the core statements in the Agile manifesto is to embrace change, even late in the development cycle. In addition, self-organization does not mean self-leadership. The lack of leading the developers and the testers to success is causing here most of the problems. Most probably Magnus' view here is shaded due to some emotional reaction to some uncomfortable perception. Therefore he's not seeing the underlying problem here. I don't claim that Magnus should be a Vulcanian, but he should at least know how to use his emotional reactions wisely.

Tim: (pauses) *"I'm not sure I'm available on Sa..."*

Magnus: *"You keep saying that. You've said that every week for the last eight weeks, and yet you've still managed to come in. It's not like this should be a surprise. The CFO said we had to ship by the end of the quarter, Sales wanted all these features for the fall, Andy wanted that API put in for that thing he's working on, and Support wanted everything fixed from the last version—now that one was a disaster; bad luck, mostly. Anyways. You've known right from the beginning that the schedule was really tight; that's what we've been saying since day one. Everybody agreed that failure wasn't an option, so we'd need maximum commitment from everyone all the way. Listen, Tim, you're basically a good guy, but quite frankly, I'm a little dismayed by your negative attitude. That's exactly the sort of stuff that brings everybody down. This is supposed to be a can-do organization."*

Tim: *"Okay. I'll come in."*

The aforementioned flaws boil down in Magnus' last statement here. He does not know how to give in to the pressure from multiple sources; he does not know how to keep his emotion out of the way to do proper management. Tim is just reacting here to the clear blaming behaviour from Magnus, and maybe this is the single alternative he now has in this conversation – of course, besides leaving the company.

What I'm really wondering now is, whether Magnus had some education in managing software projects, or if all his reactions were self-educated. What about the manager(s) in your company? What is the situation that you are facing? Do you find yourself in the position of Tim? How would you react to Magnus' statements? What would be the outcome of that conversation for you?

Biography



Markus Gärtner studied computer sciences until 2005. He published his diploma thesis on hand-gesture detection in 2007 as a book.

In 2010 he joined it-agile GmbH, Hamburg, Germany, after having been a testing group leader for three years at Orga Systems GmbH. Markus is the co-founder of the European chapter in Weekend Testing, a black-belt instructor in the Miagi-Do school of Software Testing, contributes to the ATDD-Patterns writing community as well as the Software Craftsmanship movement. Markus regularly presents at Agile and testing conferences, as well as dedicating himself to writing about testing, foremost in an Agile context.

Markus can be contacted on Twitter @mgaertne

Feedback & Responses: April'11 Issue

I'm proud to be read in association with such terrific authors. And I LOVE the artwork, throughout! I have some comments on the articles.

Lanette's article is terrific, because it says what needs to be said & nobody else is saying. We need more participation, & we also need more conferences where the participants really get to participate, rather than simple sit in rows and listen to somebody's powerpoint lecture.

Adam Yuret's article is again, right on topic. I can't tell you how many times I've heard the assumptions he lays open to inspection and rebuttal. A super job!

Brad Swanson's article deserves the same comments as above. I love to read an author telling it like it is, not hiding behind the usual myths that "we are doing Agile Testing." Really terrific, from a guy who has obviously been in the trenches.

I think **Shmuel Gershon's** article does an excellent job of detailing all sorts of ways you shouldn't evaluate testers, and he also explains what a hard job it is to do properly. But I think the article misses the main point: Why are we evaluating individual testers? Especially in an issue that emphasizes the team-nature of properly done testing? The article we want is "How do you evaluate testing teams?" No, wait, after what we know about Agile teas, why are we evaluating testers separately from other team components? Sorry, but this is a sore point with me.

Martin Jansson's article is also excellent. His one big example is good, but I'd like to see at least one example give support for each lecture point.

Joel's article is good where it's good, but I disagree on a number of points, particularly that I don't think testers should be prioritizing bugs (severity and criticality). I think I cover this in my book, Perfect Software. It's just not the tester's job, mostly because the tester does not have the info to support these ratings

The Scrum Primer was as far as I've gotten so far, but as usual, it does an excellent teaching job. I'm hoping the authors will soon assemble all these articles into a book.

So that's it for the present. I hope to review Tea-Time on my blog soon. The world needs to know about it.

Thanks for the super job you're doing!

- Jerry M. Weinberg

Hi Jerry,

We are honored that you consider Tea-Time worth reviewing on your blog. The greatest honor we have had yet. We also thank you for your time given to review our April Issue.

- Editor.

The Joke's on You !



by Nathalie Rooseboom de Vries

I recently had to travel and when browsing through the bookstore on the airport, I stumbled upon the book 'Plato and Platypus walk into a bar'. It's a small book, where philosophic principles are explained through jokes (www.platoandplatypus.com). I just kept grinning throughout the whole book, right up to the glossary where even the definitions are explained with humor and a timeline where it is stated that at a certain time Occam invented the Gillette Mach 3 (Occam's razor).

So what has this to do with testing? Well actually nothing directly, but the book inspired me to think of jokes that might spice up my lessons to learn others about testing or to explain things to my customer. Mind though; jokes aren't appropriate at every moment so use them wisely and only if the situation is of such that a joke can be used.

So I ran through the book again and noted the jokes there and thought of them where they might come useful during testing. I haven't written down a whole explanation at each joke to specifically highlight its use, cause I think you can think of the situations yourself and that makes good food-for-thought. Enjoy...

"Is that defect (or behavior) truly caused by that action?"

Sometimes you have two similar outcomes that might let you think that the causes are the same (in Philosophy this is called 'Argument from Analogy'). So here's the joke that shows that this isn't necessarily true:

A ninety-year old man went to the doctor and said, "Doctor; my eighteen-year-old wife is expecting a baby."

The doctor said, "Let me tell you a story. A man went hunting, but instead of a gun, he picked up an umbrella, shot the bear and killed it."

The man said, "Impossible. Someone else must have shot that bear."

The doctor said, "My point exactly!"

"Jumping to conclusions"

There are situations that tempt us to jump to conclusions, specifically when doing root cause analysis. Just because something might seem that way doesn't mean it is that way, sometimes the cause is something that might seem highly unlikely or illogical to us.

"An Irishman walks into a Dublin bar, orders three pints of Guinness and drinks them down, taking a sip from one, then a sip from the next, until they're gone. He then orders three more. The bartender says, "You know, they'd be less likely to go flat if you bought them one at a time".



The man says, "Yeah, I know, but I have two brothers, one in the States, one in Australia. When all went our separate way, we promised each other that we'd all drink this way in memory of the days when we drank together. Each of these is for one of my brothers and the third one is for me."

The bartender is touched, and says, "What a great custom!"

The Irishman becomes a regular in the bar and always orders the same way.

One day he comes in and orders two pints. The other regulars notice, and a silence falls over the bar. When he comes to the bar for his second round, the bartender says, "Please accept my condolences pal."

The Irishman says, "Oh no, everyone is fine. I just joined the Mormon Church, and I had to quit drinking".

I could just go on and on with citing jokes and relate them to situations that might occur during our work as tester, but I encourage you to find them yourselves and make a collection that you can use directly when the situation calls for it. I think that sometimes a joke can be more effective than any other lengthy explanation and most of the times the message will also stick.

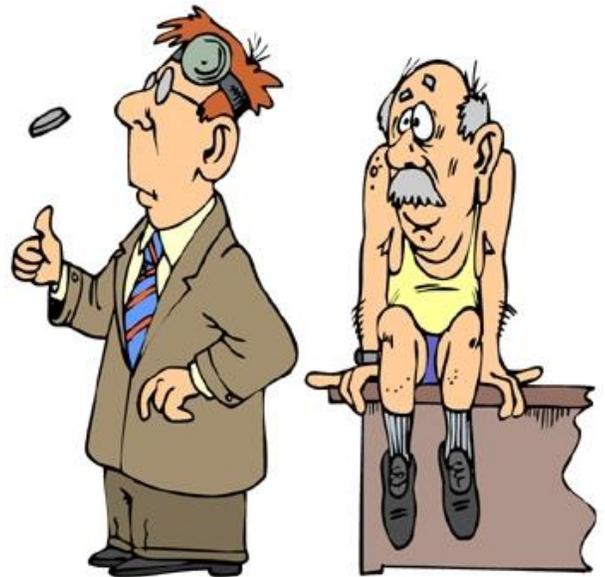
With regards to the last remark about lengthy explanations I have one final joke, also about (inductive) logic and keeping things simple.

It doesn't have to be complicated...

Holmes and Watson are on a camping trip. In the middle of the night Holmes wakes up and gives Dr. Watson a nudge.

"Watson," he says, "look up in the sky and tell me what you see".

"I see a million of stars, Holmes." Says Watson.



"And what do you conclude from that, Watson?"

Watson thinks for a moment. "Well," he says, "astronomically, it tells me that there are millions of galaxies and potentially billions of planets. Astronomically, I observe that Saturn is in Leo. Horologically, I deduce that the time is approximately a quarter past three. Meteorologically, I suspect that we will have a beautiful day tomorrow. Theologically, I see that God is all-powerful, and we are small and insignificant. Uh, what does it tell you, Holmes?"



"Watson, you idiot! Someone has stolen our tent!"

Okay, I said I had only one joke for you. But when looking into the book again, I saw the next two, that might benefit you. The first is all about asking the right questions, which are an essential part of our expertise of software testing, but also might learn your client something when making a (change)request. The second one is a good one when thinking about the context somebody is thinking in and that it might not be the same as yours, that happens when one sentence can have two meanings...

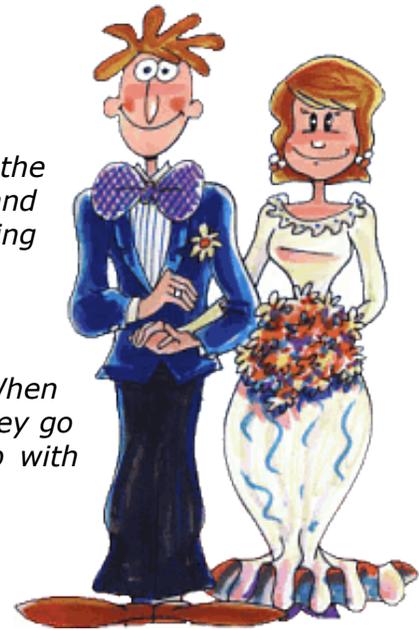
What is the question again?

A young married couple moves into a new apartment and decides to repaper the dining room. They call on a neighbor who has a dining room the same size and ask, "how many rolls of wallpaper did you buy when you papered your dining room?"

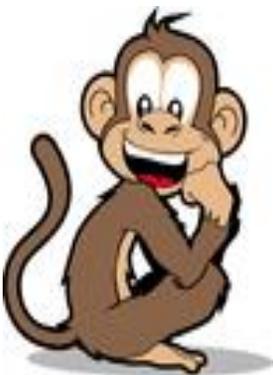
"Seven" he says.

So the couple buys seven rolls of expensive paper, and they start papering. When they get to the end of the fourth roll, the dining room is finished. Annoyed, they go back to the neighbor and say, "We followed your advice, but we ended up with three extra rolls!"

"So," he says, "that happened to you too..."



That can mean two things...



In a bar is a piano player with a monkey that goes around after each number collecting tips. While the piano player is playing, the monkey jumps up on the bar, walks up to a customer, and squats over his drink, putting his testicles in the drink. The man is miffed, walks up to the piano player, and says, "Do you know your monkey dipped his balls in my martini?"

The piano player says, "No man, but hum a few bars, and I can probably pick it up"

Sometimes a small **Tea-time** offers you something that definitely makes others' time worth enjoying. I hope you had a nice Tea-time while reading this. ☺

Biography



Nathalie has been in working at Capgemini since November 2004. She fulfills various roles within the Capgemini organization on the Software Testing field of expertise, being an ExpertGroupLeader of Testing Technologies and Processes, the CoP Testing leader and Software Testing (and Quality) advisor in the Dutch CTO office. Besides these internal roles she also is active in the (inter)national Software Testing Community, where she is a regular attendee and speaker at various conferences and publicist in various magazines.

She is actively involved in the development of the new international standard (ISO 29119) for Software Testing via the Dutch NEN committee for Software & System Development.

Nathalie can be contacted via her e-mail id funtestic.fanatic@gmail.com or on Twitter [@FunTESTic](https://twitter.com/FunTESTic)

Haven't you casted your Vote yet ?
Join the Heat and Make the Difference.
Your single Vote will make the difference for millions of others.

Teach-Testing →

Click here to Cast Your Vote





In the school of Testing

for your better learning & sharing experience

What Holds for you in the Cloud ? (Part 1)

by Dr . Meeta Prakash

This is going to be series of articles that I'd be writing for Tea Time with Testers. The idea is to get some more awareness on the 'cloud' which is becoming the buzz word for tomorrow. And if this is the space where development is headed to, how can we testers be far behind.

Reminiscing history; 'Cloud' is a term that has mesmerized me since my childhood. How could water vapor converge to something so beautiful ?? It used to be my favorite pastime to watch the clouds be blown about with wind making interesting shapes that I tried to identify with my childish imagination mapping to various known things to me like shapes, animals etc.

With age, it was a fascination to dream if angels actually lived in clouds Or there was any truth in Enid Blyton stories when she described a beautiful world up above the cloud with candies, pixies and adventures in her novels .

Growing up, another big fun was to know how would the colors and feel of cloud impact the weather and our day today life. But for each of these questions my first seeking point was my grandfather, parents or other elder family members.....

Today, as I grow up in the role of a professional in IT field, I also realize that there is a new meaning to the term 'cloud' in my life.... This a term that started picking up few years back and more I heard of it, more it intrigued me. And when I set out to seek an answer my first seeking point was google that instantly lead to my next favorite Wikipedia ☺

And lo-behold ...I was introduced to the world of cloud computing in 5th generation of computing. It also made me go back to the nostalgic days of my learning on VM.

It was from Wikipedia that I learnt that "The actual term "cloud" borrows from telephony in that telecommunications companies, who until the 1990s primarily offered dedicated point-to-point data circuits, began offering Virtual Private Network (VPN) services with comparable quality of service but at a much lower cost. By switching traffic to balance utilization as they saw fit, they were able to utilize their overall network bandwidth more effectively. The cloud symbol was used to denote the demarcation point between that which was the responsibility of the provider, and that which was the responsibility of the user. Cloud computing extends this boundary to cover servers as well as the network infrastructure. The first scholarly use of the term "cloud computing" was in a 1997 lecture by Ramnath Chellappa.

And I was WOW!! I like it ...another Indian to get credit for a discovery ☺

When cloud was initially introduced, the concept was around making the customers free from nuances of so many things that have to be dealt with on Infrastructure and environment part (like servers, networks, services, storage devices and applications) from the end user perspective.

In layman's understanding world, "cloud" can be used to describe the infrastructure and the network which are not visible to the end user but provide service as needed by the user. It is like a big black-box in the network domain.

Wikipedia further describes it as

Cloud computing describes a new supplement, consumption, and delivery model for IT services based on Internet protocols, and it typically involves provisioning of dynamically scalable and often virtualized resources. It is a byproduct and consequence of the ease-of-access to remote computing sites provided by the Internet. This frequently takes the form of web-based tools or applications that users can access and use through a web browser as if they were programs installed locally on their own computers.

Typical cloud computing providers deliver common business applications online that are accessed from another Web service or software like a Web browser, while the software and data are stored on servers.

Why are we, the IT industry drawn to the cloud has been primarily coz the ROI benefit is immense to be ignored. They also help provide us with related advantages like

1. Benefit on speed, complexity and cost
2. No investment required on additional hardware for storage, data management and servers etc
3. Providing independence to use the applications from anywhere and at anytime
4. Allowing to reach far and wide with internet connectivity

The cloud mainly comprises of 5 layers on an IP covering

1. Client
2. Application
3. Platform
4. Infrastructure
5. Server

You can use the **link** to read more on Wikipedia on cloud and its basics.

The first technique to get popularized on cloud computing was SAAS (Software As A Service). In this case the software is placed in some central place and not hosted on local machines. It is shared with users virtually based on their needs.

The next popular thing to be introduced was PAAS (Platform As A Service). In this case it was not only the software but also centralizing the end user's platform on which they were working.

Further to this we were also introduced to IAAS (Infrastructure As A Service) and CAAS (Communications As a Service) where the Infrastructure and the communication were centralized for the end users.

While the cloud has a lot of direct tangible benefits, there are multiple risks associated with it to. These risks have a high impact testing dependency. Some of the key risks can be around the following areas

- Downtime of the environment
- Technology change on cloud that can impact the solution which is hosted
- Control and safety of the data hosted on the cloud
- Security of sensitive information transfer
- Compatibility issues

I'll end this article here and my next article will talk more into the intricacies of cloud which further we'll be exploring into testing nuances.

Biography



Dr. Meeta Prakash is a passionate tester who strongly believes in leadership by example. She is a PhD in software testability. Has 12+ years of experience in Software Testing and Automation. Has worked on multiple domains and technologies across product and service industry in IT. She has extensive experience in program management, project management, people management and defect handling. She also has experience in the Global Delivery Model of IT Industry, Offshore Development Centre and the Build Operate Transfer Model.

Test techniques and test process improvement are her key interest areas. She is trained by some of the best teachers in software testing industry and is an active member of international testing community.

Meeta blogs at testingthetestable.blogspot.com & can be reached on Twitter @[meetaprakash](https://twitter.com/meetaprakash)



Claim ur S.T.O.M. Award.

Jump on to Page 59

Test Cases in Agile A Waste of Time?



by Petteri Lyytinen

Introduction

There are a lot of discussions related to testing and quality assurance in the LinkedIn discussion groups. Every now and then a topic pops up that attracts a lot of attention, plenty of comments and even causes some heated debates. One such topic was the matter of test cases in agile(1). That particular discussion thread went on for close to a year before showing any signs of fading. There were a lot of good comments in the thread and I thought it would be nice to share some thoughts of the discussion with people who were not aware of it. So when I was asked to write an article of a topic of my choosing in this magazine I decided to share those thoughts here instead of blogging them.

Many of the more heated comments in the discussion were a result of a communication gap that was, at least partially, caused by a very clear distinction between people from different schools of software

testing. Personally, I associate myself strongly with the context-driven school of testing (though I do not claim to speak for the school itself).

So, what is a test case anyway?

An interesting aspect of the discussion was that there were a lot of people talking about test cases but there was no clear consensus of what constitutes a test case. This, obviously, was a cause for some misunderstandings and heated comments since people were speaking in the same terms but talking about entirely different things.

Some people stated that a test case refers to a predetermined sequence of steps to execute in set order, with exact inputs to use and with expected results for each step. Others defined a test case as a set of instructions to execute but leaving the testers with some room for exploration by not stating exact inputs to be used. Yet others stated that all testing should be automated as much as possible and a test case simply refers to a specific automation script.

Personally, I find these definitions lacking. They may be repeatable but running through the exact same motions every time is unlikely to reveal anything new. Additionally, it is mind-numbingly boring and a bored tester is more prone to making errors.

In my view, one of the main purposes of testing is to keep looking for new ways how an application might misbehave. After all, our number one deliverable is information about the state of the application so that the team can either proceed confidently or take corrective measures to address discovered issues. Later on, the same information we testers help provide will be needed by the management or a customer so they can make a well-informed, responsible decision about shipping. You can not achieve that with strict test cases that allow no deviation – unless you increase the number of such test cases to absurdity minus one to account for as many scenarios and exceptions as possible.

Also, you can not automate intuition.

The definition I use is that **a test case is a representation of an idea for a test**. These ideas can then be used to support the actual testing depending on what tests are the most valuable or the most important at any given point in time. There are no predetermined scripts so the tests can be freely adjusted when a project's needs and requirements invariably change over time. Also, there are no limitations as to where this approach can be used. You can use the ideas to support testing of a specific functionality, the user interface, or the integrity of data transmitted via some underlying communication protocol.

In brief: The idea is a guideline – it is a support that the tester can use to execute an actual test. The ideas help the tester get the big picture before drilling down to the specifics in practice. More about the practical implementations next.

Documentation in agile

One of the principles behind the agile manifesto says: "Simplicity – the art of maximizing the amount of work not done – is essential". This is also in line with lean principles where the aim is to minimize the amount of unnecessary work and maximize the amount of value-adding work. Some people in

the discussion interpreted these to mean that test cases are not needed at all if your project model of choice is any one of the agile variants around.

I disagree.

No agile method I am aware of encourages you to drop out documentation entirely but to simply use it more effectively and for the right reasons, in the right place, at the right time. In my view, test cases work as an excellent form of documentation of testing in agile. Especially when they are written when needed and only contain just enough information to be useful. That is not enough, however. At least personally, I have found it helpful if I also know where and what kind of issues an application has had in the past. Bugs tend to cluster and sometimes old issues reappear, even after a long time. So, couple the test cases with a bug tracking system and you will also have historical data supporting your testing – provided the bug tracking system is used in a responsible, consistent manner.

Other people in the discussion stated that regardless of project model you will need an exhaustive set of test cases with exact steps to follow, inputs to use and results to expect.

I disagree again.

Not only does that claim directly contradict the agile manifesto (“working software over comprehensive documentation”), it also contradicts lean principles. You would be creating a lot of waste in the form of an inventory of test cases that are not needed right now and with too much detailed information in them that will rapidly deteriorate. Requirements change so the more detailed test cases you have the faster and more often the details in them will become outdated. As a result, the amount of non-value adding effort required increases – which is another form of waste in lean.

Based on the above, my personal approach is something that I call Iterative Test Development. It means that when I read through a user story/use case or some specification, depending on the project at hand, I write down some simple one-liners on what I think needs to be taken into account when testing. These are my ideas for tests. I also often briefly discuss with other team members to see if they can come up with an idea or two I may have missed.

After that, I will fill in the minimum required amount of details needed for each test case. That may mean giving a URL to a web page, stating some specific constraint set by the application, or anything else that a tester absolutely has to be aware of in order to be able to perform testing of that idea.

Summary

Agile models are widely misunderstood and often consciously abused for a number of reasons. Testing in an agile environment is no exception. So, it is no wonder that many people are confused. In my opinion, the discussion was a good proof of that. Not many of them go on actively for almost a year.

Project models and testing methodologies may change but the ideas remain and my view is that test cases have not become obsolete just because software is developed in a different way but it is likely more effective to write them in the agile spirit as well. The point is to do it but not overdo it.

While I realize, and readily admit, that this article has a strong bias towards the context-driven school of testing, my point was not to go through all of the views presented in the discussion but to share my thoughts inspired by it. Hopefully, I have given you some food for thought.

For anyone interested, I have included a direct link to the discussion below. It can be found under the group "Software Testing & Quality Assurance".

Also, should you have any comments or questions regarding this article, I would be happy to discuss them.

Reference

[1]:
<http://www.linkedin.com/groupItem?view=&gid=55636&type=member&item=18633715>

Feedback & Responses: April'11 Issue

Wow! I Am so Humbled! It makes me so proud to see my article printed with such wonderful, trusted testers and authors who I've admired for years. In the same issue to also be included with some of the newest bold writers in the industry, like Adam Yuret is great fun!

- Lanette Creamer

Hi Lanette,

It's our pleasure that you enjoyed writing a fresh article for Tea-time with Testers. Thank You.

- Editor.

Great work guys! And for me at the right time. My organization is going to implement Agile and the Scrum . Both articles are really helpful!

- Rajeshwari Yedravkar

Hi Rajeshwari,

We are glad to know that we could be of indirect help. Thanks for writing us. Keep reading Tea-time© .

- Editor

Biography



Petteri Lyytinen works as a Test Manager in a small Finnish IT service company called Uoma Oy.

He is a co-author of an upcoming book titled "How to Reduce the Cost of Software Testing", a conference speaker, blogger and a member of the Miagi-Do School of Software Testing. He was also nominated as a candidate for Finnish "Tester of the Year 2011".

Petteri can be reached via LinkedIn or Twitter, under the name **@plyytinen**.



Find us on
Facebook

testing intelligence

- its all about becoming an intelligent tester



an exclusive series by **Joel Montvelisky**

A Good Tester Asks Good Questions!

Testing a new application or product is always challenging. There's the challenge of working with new technologies, new languages and even new platforms; forcing us to investigate and learn how to cope with these new paradigms. But in a sense all these technical issues can be easily solved by turning to the best-friend of most geeks (like me).

Chances are that if you are asking yourself what tools to use in order to automatically test a new technology, how to write tests for a specific scripting language, or what are the weak spots on an operating system or mobile platform; the answers will already be available on the Internet, written by some early adopter or tester who took on the challenge and posted a paper or blog explaining what he did and how.



But the real challenge is coping with the things that cannot be searched on the Internet, and the ones you will need to figure out by yourself. I am referring to the functional and business aspects of your new application and the way your users will work and interact with your product once it is released.

Who do I talk to in order to get information?

The first question to ask is **"Who do you need to talk to in order understand what to test?"** and the answer, at least in the beginning of the process, is simple – TALK TO EVERYONE YOU CAN.

When you start a new project or when you're asked to test a new application you need to start by mapping all the people who can provide you with information. So make a list of everyone you can talk to – Marketing, Development, Sales, Support, the CEO – and try to talk to each of them in order to understand what you need to test (and why)!!!???

Even if you don't manage to talk to all of them (sometimes it can be tricky to get a 30 min session with your CEO...) at least try to make the list and cover as many different people as possible in order to get different perspectives and testing inputs.

What questions to ask?

Here is the hard part of the problem...

Imagine the following scenario:

You finally schedule the meeting with your CEO in order to get her inputs into what should be tested in the system. You arrive at the meeting and ask her: "Mrs. CEO, I have been given the task of testing the new product, and I wanted to ask you what do you think should be tested?". She stops and looks at you for a couple of seconds before answering: "Well, I don't know, aren't you the testing expert here...? I guess you better test everything, right? We don't want any bugs slipping out the door, do we?"

What's wrong with the scenario above? Well, basically we came to the person and asked him the wrong questions...

If you pay attention to what the CEO said she was right, we are the testing experts. She expects you to come up with what should be tested in the system, but on the other hand she is also living under the illusion that everything should or even can be tested (something we all know is impossible and even when possible, not economical in the long run).

The problem here is that we asked someone else to do our work for us, to come up with what to test, instead of asking for the input we need to understand by ourselves what needs to be tested and how. So going back to our CEO meeting scenario, what questions would have been valuable to ask? Well, you need to ask for her inputs on the system, without even talking about the testing operations. Try to understand what areas are important from a user perspective, or based on our competitive advantages, or based on what makes our application unique, etc. You should focus your questions on what is important for them.

Here are a couple of examples you can ask your CEO or VP Marketing or even Director of Sales:

- **What are the most important aspects of our Product? The things that make us unique?**

- What areas in the product are the most widely used by our customers? What type of things would make our users angry and make them choose not to work with our product?
- Where is the market focusing on today?
- How are we better than our competitors? What areas are the ones that are the most problematic in our competitors, the same areas where we want to exceed?
- Are there any risks you think we should be especially aware off? Risks in our technology, risks in the product?

Notice that we didn't ask them what to test, but we did ask what's important in their eyes (based on their experience).

In the case of the CEO, Marketing and Sales functions we will want to talk about stuff that relates to the functionality of the product. If we were to talk to our Support Team we would ask them questions related to the areas in which our users find bugs, focusing both on the places where there are the largest amount of bugs as well as where the most critical issues are found.

Finally, when talking to our development peers we will ask them about technological risks, as well as places where they are making the most changes, or where the product is relatively weak or complex and so where we should be putting more testing efforts.

The art of listening (and putting together the puzzle)

So what do you do with all the information? Basically you need to take the stuff you got, and process it in order to get a 360-degree reading of your application. What do I mean by 360-degrees? I mean from all the different angles that matter: Technology, Usability, Supportability, Competitiveness, etc.

After all, your work is to test and to provide visibility into whether the application is meeting the quality standards of your users and stakeholders. The only way to do this is by understanding what's important to all of them and creating a test plan (or work plan!) that will effectively cover it.



Biography



Joel Montvelisky is a tester and test manager with over 14 years of experience in the field.

He's worked in companies ranging from small Internet Start-Ups and all the way to large multinational corporations, including Mercury Interactive (currently HP Software) where he managed the QA for TestDirector/Quality Center, QTP, WinRunner, and additional products in the Testing Area.

Today Joel is the Solution and Methodology Architect at PractiTest, a new Lightweight Enterprise Test Management Platform.

He also imparts short training and consulting sessions, and is one of the chief editors of ThinkTesting - a Hebrew Testing Magazine.

Joel publishes a blog under - <http://qablog.practitest.com> and regularly tweets as ioelmonte



THE SCRUM PRIMER PART 4

by Pete Deemer, Gabrielle Benefield, Craig Larman, Bas Vodde

Readers are encouraged to read the first 3 parts of this series in our Previous Issues – Editor

Starting the Next Sprint

Following the Sprint Review, the Product Owner may update the Product Backlog with any new insight. At this point, the Product Owner and Team are ready to begin another Sprint cycle. There is no down time between Sprints – Teams normally go from a Sprint Retrospective one afternoon into the next Sprint Planning the following morning (or after the weekend).

One of the principles of agile development is “sustainable pace”, and only by working regular hours at a reasonable level can Teams continue this cycle indefinitely.

Release Sprint

The perfection vision of Scrum is that the product is potentially shippable at the end of each Sprint, which implies there is no wrap up work required, such as testing or documentation. The implication is that *everything* is completely *finished* every Sprint; that you could actually ship it or deploy it immediately after the Sprint Review. This means that each increment is complete slice of the final product and gives complete transparency to the Product Owner and stakeholders. They know exactly where they are at the end of every Sprint.

However, many organizations have weak development practices, tools and infrastructure and cannot achieve this perfection vision, or there are other extenuating circumstances (such as, “the machine broke”). In this case, there will be some remaining work, such as final production environment integration testing, and so there will be the need for a “Release Sprint” to handle this remaining work.

Note that the need for a Release Sprint is a sign of some weakness; the ideal is that it is not required. When necessary, Sprints continue until the Product Owner decides the product is almost ready for release, at which point there will be a Release Sprint to prepare for launch. If the Team has followed good development practices, with continuous refactoring and integration, and effective testing during each Sprint, there should be little pre-release stabilization or other wrap-up work required.

Release Planning & Initial Product Backlog Refinement

A question that is sometimes asked is how, in an iterative model, long-term release planning can be done. There are two cases to consider: (1) a new product in its first release, and (2) an existing product in a later release. In the case of a new product, or *an existing product just adopting Scrum*, there is the need to do initial Product Backlog refinement before the first Sprint, where the Product Owner and Team shape a proper Scrum Product Backlog. This could take a few days or a week, and involves a vision workshop, some detailed requirements analysis, and estimation of all the items identified for the first release.

Surprisingly in Scrum, in the case of an established product with an established Product Backlog, there should not be the need for any special or extensive release planning for the next release. Why? Because the Product Owner and Team should be doing Product Backlog refinement every Sprint (five or ten percent of each Sprint), continuously preparing for the future. This *continuous product development* mode obviates the need for the dramatic punctuated prepare-execute-conclude stages one sees in traditional sequential life cycle development. During an initial Product Backlog refinement workshop and during the continuous backlog refinement each Sprint, the Team and Product Owner will do release planning, refining the estimates, priorities, and content as they learn.

Some releases are date-driven; for example: "We will release version 2.0 of our project at a trade-show on November 10." In this situation, the Team will complete as many Sprints (and build as many features) as is possible in the time available. Other products require certain features to be built before they can be called complete and the product will not launch until these requirements are satisfied, however long that takes. Since Scrum emphasizes producing potentially shippable code each Sprint, the Product Owner may choose to start doing interim releases, to allow the customer to reap the benefits of completed work sooner. Since they cannot possibly know everything up front, the focus is on creating and refining a plan to give the release broad direction, and clarify how tradeoff decisions will be made (scope versus schedule, for example). Think of this as the roadmap guiding you towards your final destination; which exact roads you take and the decisions you make during the journey may be determined en route.

Most Product Owners choose one release approach. For example, they will decide a release date, and will work with the Team to estimate the Release Backlog items that can be completed by that date. In situations where a "fixed price / fixed date / fixed deliverable" commitment is required – for example, contract development – one or more of those parameters must have a built-in buffer to allow for uncertainty and change; in this respect, Scrum is no different from other approaches.

Application or Product Focus

For applications or products – either for the market or for internal use within an organization – Scrum moves groups away from the older *project-centric* model toward a *continuous application/product development* model. There is no longer a project with a beginning, middle, and end. And hence no traditional project manager. Rather, there is simply a stable Product Owner and a long-lived self-managing Team that collaborate in an "endless" series of fixed-length Sprints, until the product or

application is retired. All necessary “project” management work is handled by the Team and the Product Owner – who is an internal business customer or from Product Management. It is not managed by an IT manager or someone from a Project Management Office.

Scrum can also be used for true *projects* that are one-time initiatives (rather than work to create or evolve long-lived applications); still, in this case the Team and Product Owner do the project management.

What if there is insufficient new work from one or more existing applications to warrant a dedicated long-lived Team for each application? In this case, a stable long-lived Team may take on items from one application in one Sprint, and then items from another in the next Sprint; in this situation the Sprints are often quite short, such as one week.

Occasionally, there is insufficient new work even for the prior solution, and the Team may take on items from *several* applications during the same Sprint; however, beware this solution as it may devolve into unproductive multitasking across multiple applications. A basic productivity theme in Scrum is for the Team to be *focused* on one product or application for one Sprint.

Common Challenges

Scrum is not only a concrete set of practices – rather, and more importantly, it is a framework that provides transparency, and a mechanism that allows “inspect and adapt”. Scrum works by making visible the dysfunction and impediments that are impacting the Product Owner and the Team’s effectiveness, so that they can be addressed. For example, the Product Owner may not really know the market, the features, or how to estimate their relative business value. Or the Team may be unskillful in effort estimation or development work. The Scrum framework will quickly reveal these weaknesses. Scrum does not solve the problems of development; it makes them painfully visible, and provides a framework for people to explore ways to resolve problems in short cycles and with small improvement experiments.

Suppose the Team fails to deliver what they committed to in the first Sprint due to poor task analysis and estimation skill. To the Team, this feels like failure. But in reality, this experience is the necessary first step toward becoming more realistic and thoughtful about its commitments. This pattern – of Scrum helping make visible dysfunction, enabling the Team to do something about it – is the basic mechanism that produces the most significant benefits that Teams using Scrum experience.

One common mistake made, when presented with a Scrum practice that is challenging, is to change Scrum. For example, Teams that have trouble delivering on their Sprint commitment might decide to make the Sprint duration extendable, so it never runs out of time – and in the process, ensure it never has to learn how to do a better job of estimating and managing its time. In this way, without coaching and the support of an experienced ScrumMaster, organizations can mutate Scrum into just a mirror image of its own weaknesses and dysfunction, and undermine the real benefit that Scrum offers: Making visible the good and the bad, and giving the organization the choice of elevating itself to a higher level.

Another common mistake is to assume that a practice is discouraged or prohibited just because Scrum does not specifically require it. For example, Scrum does not require the Product Owner to set a long-term strategy for his or her product; nor does it require engineers to seek advice from more experienced engineers about complex technical problems. Scrum leaves it to the individuals involved to make the right decision; and in most cases, both of these practices (along with many others) are well advised. Something else to be wary of is managers imposing Scrum on their Teams; Scrum is about giving a Team space and tools to manage itself, and having this dictated from above is not a recipe for success. A better approach might begin with a Team learning about Scrum from a peer or

manager, getting comprehensively educated in professional training, and then making a decision as a Team to follow the practices faithfully for a defined period; at the end of that period, the Team will evaluate its experience, and decide whether to continue.

The good news is that while the first Sprint is usually very challenging to the Team, the benefits of Scrum tend to be visible by the end of it, leading many new Scrum Teams to exclaim: "Scrum is hard, but it sure is a whole lot better than what we were doing before!"

Appendix: Terminology

Burn Down

The trend of work remaining across time in a Sprint, a Release, or a Product. The source of the raw data is the Sprint Backlog and the Product Backlog, with work remaining tracked on the vertical axis and the time periods (days of a Sprint, or Sprints) tracked on the horizontal axis.

Chicken

Someone who is interested in the project but does not have formal Scrum responsibilities and accountabilities (Team, Product Owner, ScrumMaster).

Daily Scrum

A short meeting held daily by each Team during which the Team members inspect their work, synchronize their work and progress and report and impediments to the ScrumMaster for removal. Follow-on meetings to adapt upcoming work to optimize the Sprint may occur after the Daily Scrum meetings.

Done

Complete as mutually agreed to by all parties and that conforms to an organization's standards, conventions, and guidelines. When something is reported as "done" at the Sprint Review meeting, it must conform to this agreed definition.

Estimated Work Remaining (Sprint Backlog items)

The number of hours that a Team member estimates remain to be worked on any task. This estimate is updated at the end of every day when the Sprint Backlog task is worked on. The estimate is the total estimated hours remaining, regardless of the number of people that perform the work.

Increment

Product functionality that is developed by the Team during each Sprint that is potentially shippable or of use to the Product Owner's stakeholders.

Increment of Potentially Shippable Product Functionality

A complete slice of the overall product or system that could be used by the Product Owner or stakeholders if they chose to implement it.

Sprint

An iteration, or one repeating cycle of similar work, that produces increment of product or system. No longer than one month and usually more than one week. The duration is fixed throughout the overall work and all teams working on the same system or product use the same length cycle.

Pig

Someone exercising one of the three Scrum roles (Team, Product Owner, ScrumMaster) who has made a commitment and has the authority to fulfill it.

Product Backlog

A prioritized list of requirements with estimated times to turn them into completed product functionality. Estimates are more precise the higher an item is in the Product Backlog priority. The list emerges, changing as business conditions or technology changes.

Product Backlog Item

Functional requirements, non-functional requirements, and issues, prioritized in order of importance to the business and dependencies and estimated. The precision of the estimate depends on the priority and granularity of the Product Backlog item, with the highest priority items that may be selected in the next Sprint being very granular and precise.

Product Owner

The person responsible for managing the Product Backlog so as to maximize the value of the project. The Product Owner is responsible for representing the interests of everyone with a stake in the project and its resulting product.

Scrum

Not an acronym, but mechanisms in the game of rugby for getting an out-of-play ball back into play.

ScrumMaster

The person responsible for the Scrum process, its correct implementation, and the maximization of its benefits.

Sprint Backlog

A list of tasks that defines a Team's work for a Sprint. The list emerges during the Sprint. Each task identifies those responsible for doing the work and the estimated amount of work remaining on the task on any given day during the Sprint.

Sprint Backlog Task

One of the tasks that the Team or a Team member defines as required to turn committed Product Backlog items into system functionality.

Sprint Planning meeting

A one-day meeting time boxed to eight hours (for a four week Sprint) that initiates every Sprint. The meeting is divided into two four-hour segments, each also time boxed.. During the first four hours the Product Owner presents the highest priority Product Backlog to the team. The Team and Product Owner collaborate to help the Team determine how much Product Backlog it can turn into functionality during the upcoming Sprint. The Team commits to this at the end of the first four hours. During the second four hours of the meeting, the Team plans how it will meet this commitment by designing and then detailing its work as a plan in the Sprint Backlog.

Sprint Retrospective meeting

A time boxed three-hour meeting facilitated by the ScrumMaster at which the complete Team discusses the just-concluded Sprint and determines what could be changed that might make the next Sprint more enjoyable or productive.

Sprint Review meeting

A time-boxed four hour meeting at the end of every Sprint where the Team collaborates with the Product Owner and stakeholders on what just happened in the Sprint. This usually starts with demonstration of completed Product Backlog items, a discussion of opportunities, constraints and

findings, and a discussion of what might be the best things to do next (potentially resulting in Product Backlog changes). Only completed product functionality can be demonstrated.

Stakeholder

Someone with an interest in the outcome of a project, either because they have funded it, will use it, or will be affected by it.

Team

A cross-functional group of people that is responsible for managing themselves to develop an increment of product every Sprint.

Time box

A period of time that cannot be exceeded and within which an event or meeting occurs. For example, a Daily Scrum meeting is time boxed at fifteen minutes and terminates at the end of fifteen minutes, regardless. For meetings, it might last shorter. For Sprints, it lasts exactly that length.

Biography



Pete Deemer is a founder of GoodAgile, and co-founder of the Scrum Training Institute.

Pete is an honors graduate of Harvard University, and has spent the last 22 years leading teams building products and services at global companies. Most recently he served as Vice President of Product Development for Yahoo!, where he led Yahoo's global adoption of Scrum, which grew to over 2000 developers worldwide during his tenure.

He can be reached at his mail id – petedeemer@scrumtraininginstitute.com



Gabrielle Benefield is a Certified Scrum Trainer based in London offering classes in the United Kingdom and Europe. Gabrielle has over 18 years experience building enterprise software and web products at global companies and is a founder of the Scrum Training Institute, with Jeff Sutherland, the co-creator of Scrum, Pete Deemer and Jens Ostergaard.

Gabrielle works with clients from diverse industries including banking, telecommunications, and internet.



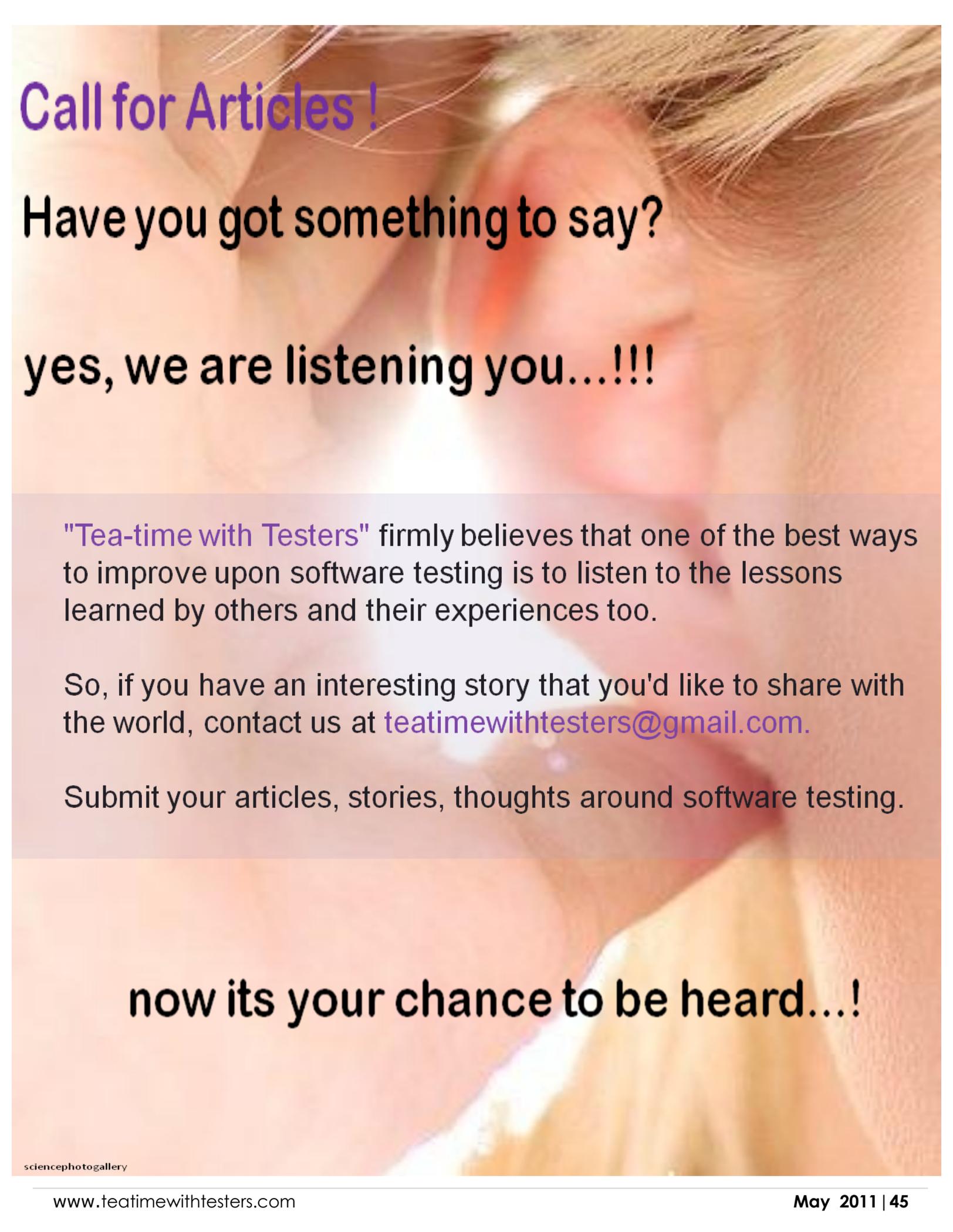
Craig Larman works as the lead coach of lean product development adoption at Xerox, and serves as a consultant for large-scale Scrum and enterprise agile adoption at Nokia and Siemens Networks (now, NSN), at Statoil and Kongsberg Maritim and Cisco-Tandberg (in Norway), at Alcatel-Lucent, and at Schlumberger and UBS, among many other clients.

He can be reached at his mail id - craig@craiglarman.com



Bas Vodde is originally from Holland, however he has lived in China, Finland, China again and currently lives in Singapore. He led the Agile transformation project at Nokia Networks and later Nokia Siemens Networks, and currently works for his own company called Odd-e. He's been working with several large products and several large company change projects.

Bas can be reached at his website www.odd-e.com



Call for Articles !

Have you got something to say?

yes, we are listening you...!!!

"Tea-time with Testers" firmly believes that one of the best ways to improve upon software testing is to listen to the lessons learned by others and their experiences too.

So, if you have an interesting story that you'd like to share with the world, contact us at teatimewithtesters@gmail.com.

Submit your articles, stories, thoughts around software testing.

now its your chance to be heard...!

T ' Talks



T. Ashok exclusively on software testing

The Diagnosis

One day Joe's roommate David was unwell. After a busy day at work, grappling with unruly software, Joe returned to his apartment and took the stairs to his pad on first floor, and set his heavy laptop back down. He opened the bag, searched for the house key and unlocked the door, welcomed by the typical darkness that he was accustomed to.

He knew the placement of the light switches very well, as he stepped to turn it on, he heard a groan. He froze, frightened by the noise; he quietly tip-toed to the light switch, turning it on rapidly. David was an owl who worked late and came in after Joe. He was shocked to see his roommate David moaning in pain.

As Joe touched David lightly, he opened his eyes lightly and said "Sorry man, it hurts, feels like I am in labour" and smiled weakly. David loved life and everybody liked his funny bone humor. Joe smiled and said "Let us go to a doctor now".

"No Joe, it was pretty bad couple of hours ago, I have taken medication and it is a lot better now. I have found a comfortable posture and would not mess with it right now. Let us go in the morning". "Ok. Do you need something?" "No, Thanks." said David as he curled up and continued to take deep breaths to alleviate the pain.

Joe left, washed up and went to the bed. He was feeling bad and slightly worried. Slowly his thoughts drifted back to office. He was getting into a new project in a domain where he had never worked before. This was a new product, he was a little lost and had trouble understanding the application. There was very little written documentation and the key folks were in a different continent. Today afternoon when it was going nowhere, he had walked into his manager's office and whined.

"Well, you have to be creative and got to come up with good questions."

Joe had played with a similar application and seemed to understand some parts of the application. But he was not able to visualize the application and its usage in its entirety. Slowly he drifted into sleep. He was awakened by sound of constant flushing in the adjacent bathroom.

"David - are you ok?" asked Joe.

"Yeah man" said David softly. "Guess something that I ate yesterday is really working out my system". As David came out, he looked pale and tired, hobbled to the nearest chair, sat down and laid his head into his cupped hands.

"Boy you look terrible, guess we better go see the doc now".

"Yeah, what time is it now?"

"8:30"

In a few minutes, they were out of the apartment, hailed a taxi and was in GoodLife hospital at 9:00.

"Hi, Is the doctor in?" Joe asked the pretty lady behind the front desk.

"No Sir, he is expected any moment. You are the first one and I will call you as soon he is in."

After a few minutes, they were ushered in to the office of Dr Holmes.

"Good morning. David, I presume you have serious pain in the abdomen with frequent vomiting in the morning" said the observant doctor surprising David thoroughly.

"Yes doctor, but how did you know my name?" David asked, not realizing that his name with #8 was displayed prominently on the back of his T Shirt.

David was an avid basketball fan and played for the local city team.



Dr. Holmes smiled and got onto the business.

"David, when did the pain commence?"

"Couple of days ago doctor, it became intense yesterday evening"

"So what do you do David?"

"I work in an ad agency as a copywriter"

"Late nights and irregular sleep?"

"Yes doctor, the day starts late and it is pretty long"

Dr. Holmes asked David to lie down and gently touched the lower part of abdomen and David winced.

As Dr. Holmes continued to ask questions on family history, what he ate yesterday, prior history of sickness, Joe was transported to a different world. A reflective person that he was, he was thinking about the problem of understanding the application.

He realized that Dr Holmes was employing a pattern of questioning that was structured yet creative. Joe knew a bit of mind mapping, and realized that the doctor was seeking information on some standard aspects like family background, lifestyle, food habits, recent activities and came up with questions when he connected these aspects. He also realized that certain answers resulted in more questions. Slowly the mist lifted and he realized that as a tester, he also needed blobs of information like customer types, types of end user, #users/type, profile of usage, key attributes, architecture, stage of development, relative ranking of the features/ users, interaction between features, feature volatility, deployment environment. It dawned on him that these blobs of information and their connections would enable him to construct a "Landscape" of the system helping him to visualize the system..

It flashed on him that good questions can be asked when these connections are attempted to be established.

"Do you have any questions David?" asked Dr. Holmes.

"YES!" said Joe emphatically as he thumped the Doctor's desk, his reverie broken. Realizing his faux-pas, he sheepishly looked at a confused David and an amused doctor.

"Guess you solved the problem. Good understanding requires an open mind, and connecting the dots" said Dr Holmes surprising Joe.

"Wow, you are a mind reader" exclaimed Joe. "Guess you are Sherlock Holmes!"

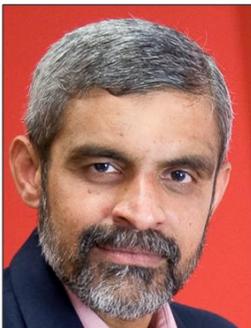
Dr Holmes smiled and said "Good understanding is like mind reading. Good Day Gentlemen".

Joe escorted a bewildered David out of the doctor's office, said "Have a great day" loudly to the pretty receptionist, winking at the elderly lady seated past the reception.

Joe knew the answer to his problem and looked forward to a lovely day at work.

P.S: Landscaping is a technique inspired by Mind mapping and is a core concept in HBT (Hypothesis Based Testing). It enables a scientific approach to questioning that aids in rapid understanding of a system. I will be talking more about this in my next article for *Tea-time with Testers'* June'11 Issue.

Note: Drop me a note at ash@stagsoftware.com or tweet me [@ash_thiru](https://twitter.com/ash_thiru) if you liked this. Thank you.



Biography

T Ashok is the Founder & CEO of STAG Software Private Limited.

Passionate about excellence, his mission is to invent technologies to deliver "clean software".

He can be reached at ash@stagsoftware.com.





Tool Watch

about various testing tool around



PractiTest

PART 2

Tea-time with Testers Rating: ★★★★★

**by Juhi Verma &
Sharmistha Priyadarshini**

Apart from the Requirement gathering, writing Test Cases & raising Bugs, we Testers also perform one equally important job i.e. Test Execution & Management in Test Management Tool.

No doubt, we all learn & adopt ourselves best to work with different Test Management Tools. But have we ever thought of something much simpler, interesting, innovative and definitely astonishing?

Well...won't it be an easy job just to have a look on Auto Generated Graphs of your Test Execution Status, Bug Statistics, Project Assignments to specific group and much more, that too just with a single sign on?

- What if your tool itself tells you that the bug you we are about to write is a duplicate of an existing bug?
- What if you can set the visibility of your project to other users?
- What if your tool is intelligent enough to handle the parent-child hierarchy of issues?
- What if your tool provides the flexibility to the views according to the user using it?

And...How about managing your Tests and Bugs from your very own I-PAD?

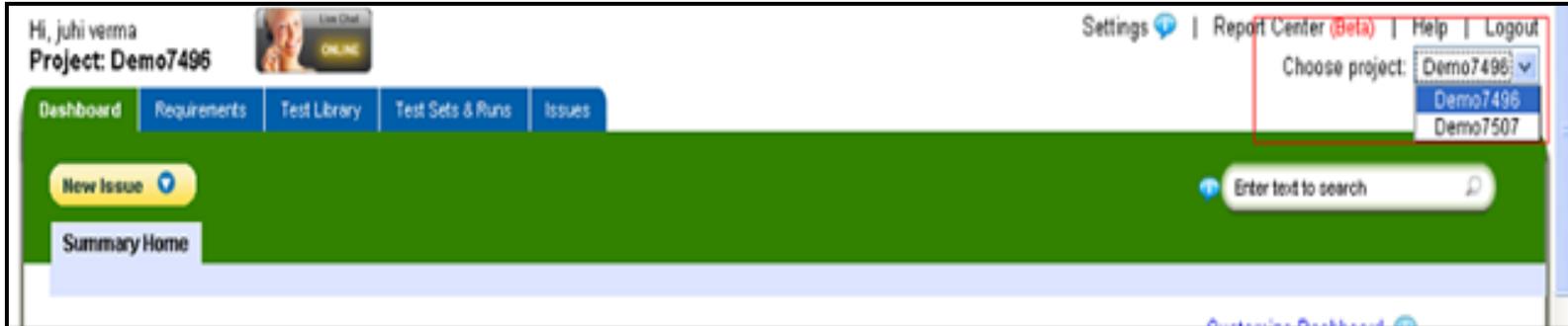
Well, PractiTest is the Tool. Let us walk you through this best Test Management Tool we have ever come across.

*We will continue from the features that we have covered in **April-2011 Issue** of **Tea-time with Testers**.*

3. Key Features of PractiTest

- **Flexibility in User Login :**

Within a single login, a provision to switch between different projects with different role (Admin/Tester/Developer) is given.



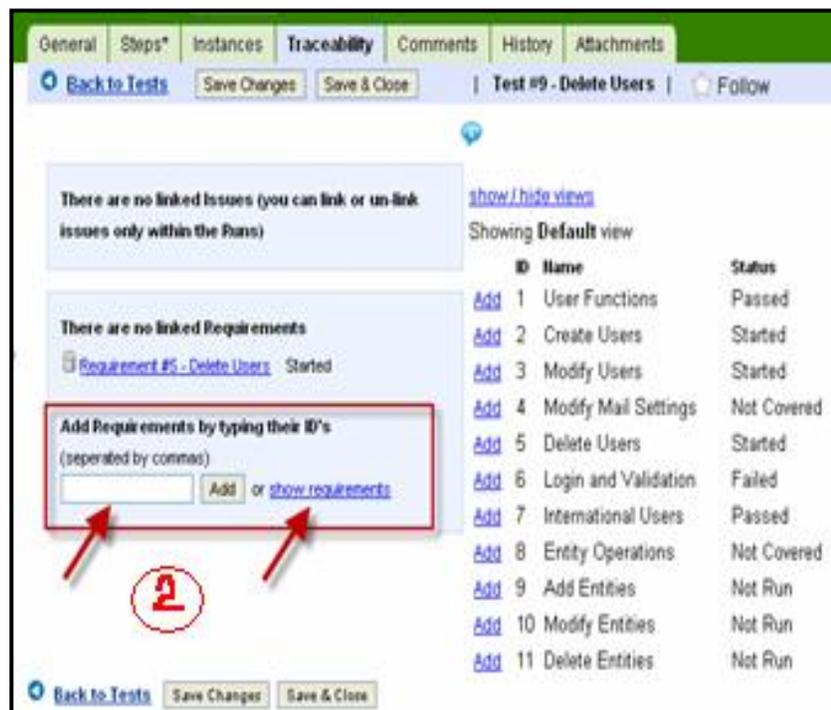
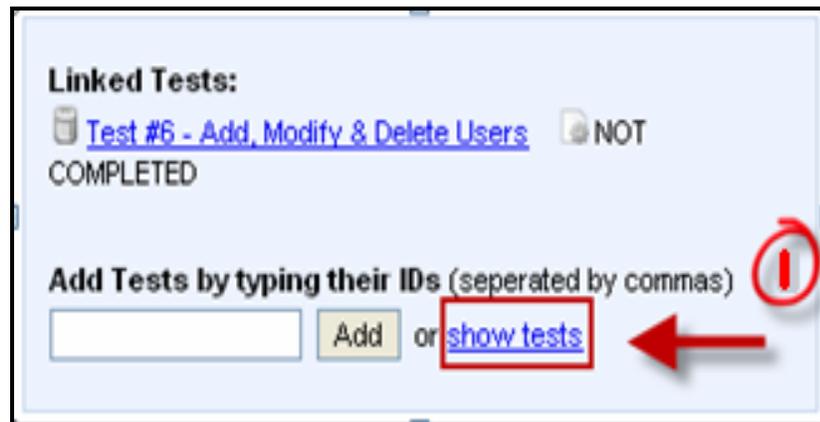
- **Traceability**

You can create traceability links to Tests in the system; this way you can keep track of your application and project's status, based on the execution and results of the tests that are linked to your requirements.

Here is how you create traceability links:

1. Go to the "Traceability" sub-tab in "Requirement" tab and choose the tests related to the requirement by either adding its ID, or clicking on the "show tests" link to choose the relevant tests from the list.
2. You can also link between tests and requirement from the tests themselves.

To do this, go to the traceability sub-tab within a test. Enter a requirement's ID or click on the "show requirements" link to choose a requirement from the list.



• **Views and cascading of views :**

Views can be used to manage all phases of your Software Testing Life Cycle, and also can customize views and filters for each application module.

To create a view (for example, for issues):

1. Go to the *Issues* tab and then click on the "Custom Views" sub tab.
2. Create a child view by clicking on the "add child view" button next to the parent view. For example, the view we create will be under "All issues". You can create a child view under any existing view.
3. Choose the View Type from the drop-down list and enter a name for your view in the field below.

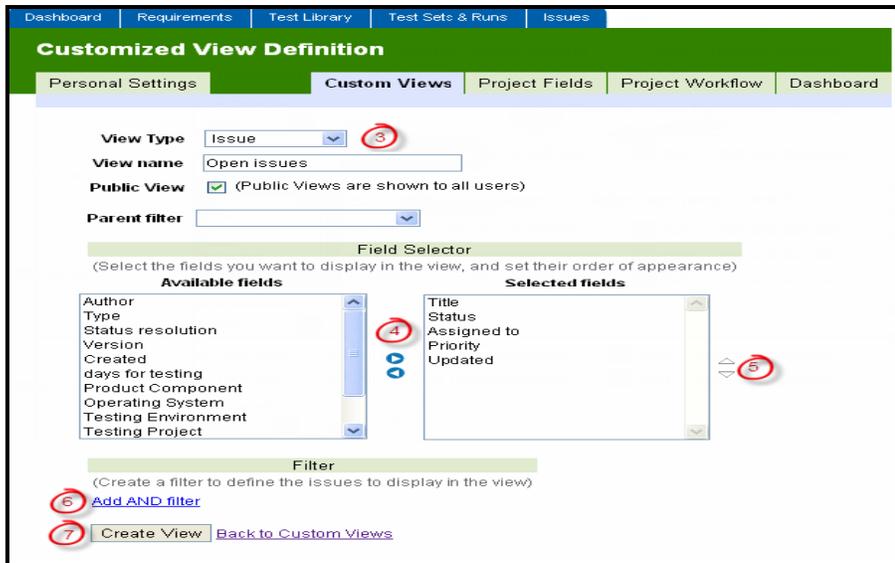
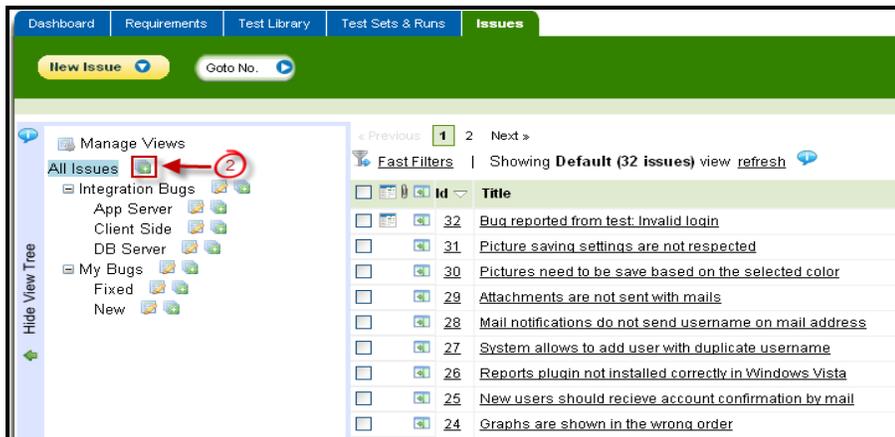
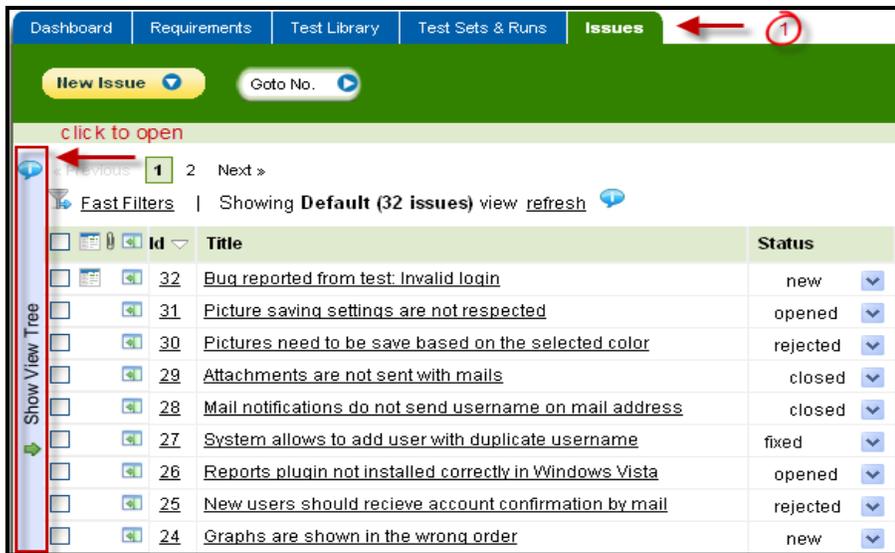
Note: The content of this field is chosen automatically when you create views from the view tree.

4. Enter a name for your view. It's best to use a descriptive name, for example, "open issues".
5. In the Field Selector section, choose the columns you want displayed on your grid by moving them to the Selected Fields box, using the blue arrow.
6. You can also rearrange the selected fields' order by pressing the arrows on the right of the Selected Fields box.

7. The filter defines the issues that will be displayed in the view. Click the "Add AND filter" link and choose the criterion you would like to filter the view by.

8. In this example, the filter will only show bugs with status "opened"
Note: you can add several criteria to your filter.

9. Finally, press the Create View button at the bottom of the page to save your view. You will be redirected to the relevant library.



Note: Views are also used in the system to create reports, dashboard entities, etc.

• **Import Test case**

One can start using PractiTest without losing any of the existing data. Simply import the requirements, tests, test steps or issues into PractiTest from any CSV file.

One can create a CSV file using MS-Excel or Open-Office. Just click on "Save As", and select "CSV".

To import your data, follow these easy steps.

1. Go to the Settings link at the top right side of your screen, and then go to the Project and Import sub-tab.
2. At the bottom of the page, you can choose which entities you would like to import into PractiTest. You can import issues, tests, test steps and requirements.
3. Select the CSV file you'd like to import the data from.
4. Map the columns in your existing file by choosing the column for each of the system or custom fields you've already created. There's no need to make changes to your CSV file!
For example, let's say this is your excel file.

	A	B	C	D	E
1	Title	description	assigned to	Status	days for testing
2	creating new users	name should be mandatory	Joan Green	passed	2
3	deleting new users	deleting not possible when special characters used	Mat Smith	passed	2
4	uploading contacts	contacts' Email address not uploaded	Joan Green	failed	1
5	downloading contacts	contacts with special characters not downloaded	Mat Smith	not complete	1

Mapping will be done according to your current columns on the CSV file. In this example, the "Title" field is in column A, the "Description" field is in column B, the "Status" field is in column D, etc.

Note:

An asterisk * denotes mandatory fields. Your imported data will be saved correctly even if these fields are empty or missing altogether; however, the next time you edit one of these issues, you will be required to select values for the mandatory fields before saving.

5. Note that in this example, the first row in the CSV file is headlines. In this case, you can check the "ignore first row" checkbox at the bottom of the page. You can also choose if you'd like to receive mail notifications of this operation for every imported entity. The default value is to disable mail notifications.

6. Click on the Import button at the bottom of the page. Once the import is completed you will be taken to a results window.

EXPORTING DATA

Export to CSV (Excel)

CSV exports are available in the requirements, test library and issues modules. Using the "Views and Reports" button in each module, you can export the information to CSV.

1. Go to the Issues tab (for example)
2. Make sure to select the view you want to export to excel by choosing it from the View Tree on the left side of the screen.

Note: this step is only relevant for Issues. In other modules, all items will be exported regardless of your current view.

3. Press the Views & Reports button & select the Export as CSV option.

Note: You will be prompted to open the file or save it to your desktop. The file generated by PractiTest is in CSV format (Comma Separated Values) and can be opened using multiple applications, such as MS-Excel.

The screenshot shows the 'Test Library' module in a web application. The navigation tabs include 'Dashboard', 'Requirements', 'Test Library' (highlighted with a red box and annotation 1), 'Test Sets & Runs', and 'Issues'. Below the tabs, there are buttons for 'New Test' and 'Goto No.' (with annotation 1). The main heading is 'Test Library: Create & manage your tests'. Below this is a table of tests. The table has columns: 'Id', 'Name', 'Status', 'Run status', 'Last run', and 'Updated'. The table contains 9 rows of test data. A 'Show View Tree' sidebar is on the left, with a red box and annotation 2 around the checkboxes. Above the table, there are controls for 'View: All Tests (9 Tests)', 'Refresh View', 'Stats' (with annotation 3), 'Clone Tests', 'Batch Edit', and 'Delete'. The table data is as follows:

Id	Name	Status	Run status	Last run	Updated
9	Delete Users	Ready	NO RUN	Never Run	05-Feb-2011 16:30
8	Delete Entities	To Repair	NO RUN	Never Run	05-Feb-2011 16:31
7	Entities and Values creation	To Repair	NO RUN	Never Run	24-Jan-2011 15:21
6	Add, Modify & Delete Users	Ready	NOT COMPLETED	24-Jan-2011	05-Feb-2011 16:31
5	User definition functions	Ready	PASSED	24-Jan-2011	24-Jan-2011 15:21
4	Login with International Characters	Ready	PASSED	24-Jan-2011	24-Jan-2011 15:21
3	Login without parameters	Draft	BLOCKED	24-Jan-2011	24-Jan-2011 15:21
2	Invalid login	Ready	FAILED	24-Jan-2011	24-Jan-2011 15:21
1	Valid login	Ready	PASSED	24-Jan-2011	24-Jan-2011 15:20

Stats

Stats are available in *test library* and *test sets & runs* tabs.

1. Go to the test library.
2. Choose the relevant data you'd like to see in the Stats, by selecting the checkboxes on the left side of the test library grid.
3. Click on the "Stats" link. You will be redirected to a new window where you can see your stats.
4. Click on the link to view the Run status numbers by "Assigned To".
5. You can also see execution trends by choosing a date range to view the relevant data for these dates.

Dashboard | Requirements | **Test Library** | Test Sets & Runs | Issues

New Test | Goto No.

Statistics for (Total of 7 Tests)

Execution statistics

Click [here](#) for Run status numbers by Assigned To **4**

Click below for date range interval:

[Last 1 day](#) | [Last 7 days](#) | [Last 2 weeks](#) | [Last month](#) **5**

Test Plan statistics

Click [here](#) to see the average and sum numbers custom fields

Dashboard | Requirements | **Test Library** | Test Sets & Runs | Issues

New Test | Goto No.

Statistics for (Total of 7 Tests)

Execution statistics

Run status by assigned to ([show / hide](#))

Run status by Assigned To	Pete Johnson	Joel Montvelisky
FAILED	0	0
BLOCKED	1	0
PASSED	1	1
NOT COMPLETED	0	1
NO RUN	1	2

Run status by Assigned To

Legend: NO RUN, NOT COMPLETED, PASSED, BLOCKED, FAILED

• Customizing your fields

Hi, Joan Green **1** → Settings | Report Center (Beta) | Help | Logout

Project: Testing Project Choose project: Testing Project

Dashboard | Requirements | Test Library | Test Sets & Runs | Issues

Settings - Project fields

Personal Settings | Custom Views | **Project Fields** | Project Workflow | Dashboard | Users & Groups | Project & Import

Name	Field Format	Linked to	Created	Modified	
Account Owner	User		24-Jan-2011	24-Jan-2011	edit delete
Browser	List	Issue, Test, Requirement	24-Jan-2011	24-Jan-2011	edit delete
days for testing	Number	Issue, Test, TestSet	05-Feb-2011	05-Feb-2011	edit delete
Fix Next Build	Checkbox	Issue	24-Jan-2011	24-Jan-2011	edit delete
Operating System	List	Issue, Instance	24-Jan-2011	24-Jan-2011	edit delete
Product Component	List	Issue, Test, Requirement	24-Jan-2011	24-Jan-2011	edit delete
Sub Component	Linked list	Test, Requirement	24-Jan-2011	24-Jan-2011	edit delete
Test Level	List	Test, TestSet	24-Jan-2011	24-Jan-2011	edit delete
Testing Environment	List	Issue	24-Jan-2011	24-Jan-2011	edit delete
Testing Phase	List	Issue, Test, TestSet	24-Jan-2011	24-Jan-2011	edit delete
Testing Project	Linked list	Issue	24-Jan-2011	24-Jan-2011	edit delete

[Create New Custom Field](#) **3**

Custom fields allow you to customize your entities (requirements, tests, test sets and issues) to match your project and process' needs.

To create a new field, for example, to create a list of all the browsers supported by your system.

1. Click on the *Settings* link at the top right side of the screen
2. Go to the *Project Fields* sub tab
3. Under "Custom Fields", press the Create New Custom Field link at the bottom of the list.

4. Enter the values for your list (for example IE, FF, Safari) and press the *Add value* link after each value.

5. Enter the name of your field and choose the value *List* from the *Field Format* drop-down list. (This is where you can choose different formats if you want to add, for example, a text field, a memo field, a checkbox, etc.)

Note: You can re-order the display of the values using the arrows next to the list, as well as delete unwanted value link.

6. Under "Field linkage to entities", select the entities you want to assign this new field to, as well as whether it is mandatory or if it has a default value. In this example, we made the browser field available for issues, tests and requirements, and we made it mandatory for issues.

7. Press the *Submit* button to be taken back to the list of Custom Fields already defined. Your new custom field has been added to the chosen entities; here you can see it was added to Issues as a mandatory field.

- **Customizing users & groups and give permissions**

Groups help organize users and control their permissions Within PractiTest.

Follow these steps to create a new group.

1. Go to the Settings link and then to the Users & Groups sub tab
2. On the right side of your screen you can add new users and create groups. Add a new user by entering their email in the "add a new user" section. Choose the group you would like to add the user to. Users are added by default to the *Testers* group.

Note:

When new users are added to PractiTest, the users receive an Email with their initial login info. However, when adding existing users to an additional project, they will not receive an Email, and will be able to see the project on their "project list" the next time they log into PractiTest, or after refreshing the screen.

3. Create a new group by entering a name for the group and clicking the *Create group* button.

4. To edit a group's permissions and users, click on the "edit" link next to the group's name. To add a user, simply choose the user's name from the dropdown list and click on the "Add" button. Use the checkboxes to determine the group's permissions.

Note: you can add a user to more than one group.

• **Customize your workflow**

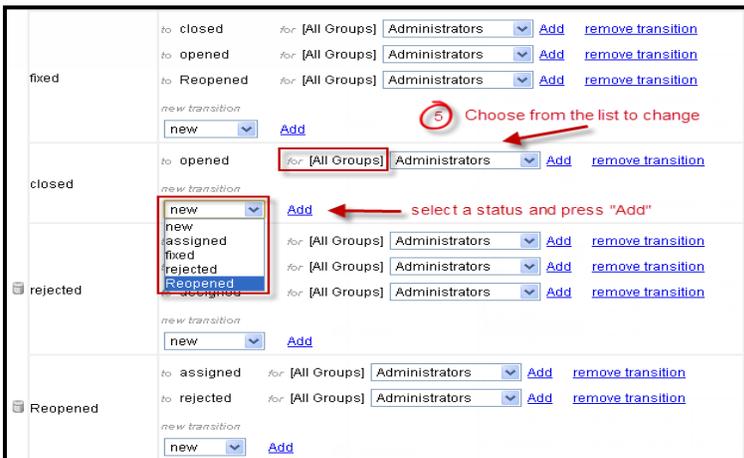
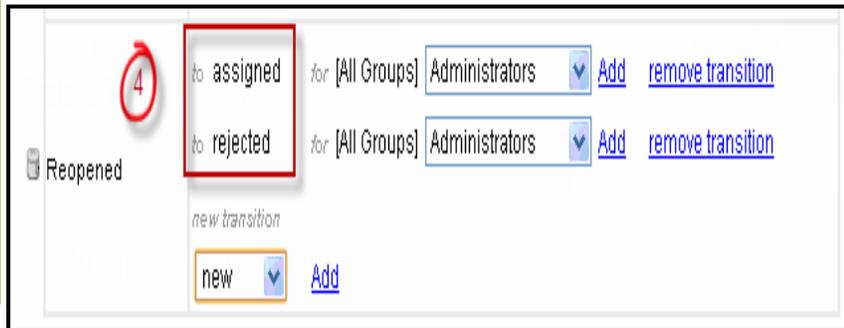
PractiTest allows to customize issues' Life-Cycle by adding statuses to the workflow or editing existing ones. For example, add a new status called "reopened", and define that issues can only go from "closed" and "fixed" to "Reopen" .

- Go to the *Settings* link & then to the *Project Workflow* sub tab.
- Select the value *issue* from the *Entity Type* drop-down list.
- Type the name of your new status (for example, 'Reopened') and press the *Add* button. The new status is added at the bottom of the list
- Define the transitions to your new status by selecting values in the appropriate status, based on your organization's Bug Lifecycle.

Note:

The stats displayed in the transitions box on the right of each status name indicate the states to which the issue can be transitioned to. Select additional states by choosing them from the drop down list and pressing the *Add* link.

For example, we will make it possible to change a status from "reopened" to "assigned" and to "rejected" only.



If we decide that an issue can go from "closed" and "fixed" to "reopened", we should add this transition to these statuses as well.

You can limit the groups that can perform the transition by selecting them from the selection box and clicking "Add". By default, each transition can be made by all groups.

In this example, the status "closed" can be changed to "opened" by ALL GROUPS. Choose a group from the dropdown list to change this default value.

To Be Continued In Next Issue

Biography



Juhi Verma is an Electronics Engineer working with Tata Consultancy Services (Mumbai) as a Tester. During her spell she has also worked as a programmer but she now prefers software Testing over programming as it's a dual fun, she says.

Testing is her passion and she enjoys participating and conducting testing related activities.

Juhi also works as a Team member at Tea-time with Testers.

She can be contacted at her personal mail id juhi_verma1@yahoo.co.in or on Twitter [@Juhi_Verma](https://twitter.com/Juhi_Verma).



Biography



Sharmistha Priyadarshini is currently working as a Test Engineer at Tata Consultancy Services (Mumbai).

Sharmistha is die hard lover of Data Base testing as she finds it challenging. Being a programmer in past she now loves software testing too as it gives her more scope for analysis.

Sharmistha can be reached via her mail id sharmistha.priyadarshini@tcs.com



Testing

PUZZLES

by Sebi

Introducing a new way of claiming your
Smart Tester of The Month Award.

Send us an answer for the Puzzle below
b4 10th June 2011 & grab your Title.

Gear up guys...

.....**Time To Tease your Testing Bone!**

Puzzle "Superstition"



This is the legend :

	2		13		31		53
	3		17		37		59
	5		19		41		61
	7		23		43		67
	11		29		47		71

Each token has a prime number associated

Solve the next 7 lines by using the operators: "+", "-", "*", "/":

10 icons = 2012

Example:

1 icon x 2 icons + 1 icon + 1 icon x 2 icons x 2 icons - 1 icon x 1 icon + 1 icon - 1 icon = 2012

Send us your answer with proper justification to teatimewithtesters@gmail.com with Sub: Testing Puzzle



Biography

Blindu Eusebiu (a.k.a. Sebi) is a tester for more than 5 years. He is currently hosting European Weekend Testing. He considers himself a context-driven follower and he is a fan of exploratory testing. He tweets as @testalways. You can find some interactive testing puzzles on his website www.testalways.com

Our Testimonials

Hi There!

My name is Mauri, a QA & Tester one-man-department fighting everyday with 5 of the spanish best Java developers. I would like to subscribe to tea time with testers ezine in order to learn and be able to incorporate new testing techniques and tools, so that I could defend myself against the tons of bugs I face each morning.

Hope to receive next issue and start my training very soon.

- Mauri Edo

Dear Mauri,

Thanks for counting on us. We shall provide you with the best we can.

-Editor

Glad to receive you ezine to stay tuned with testing related trends & discussion

- Jon

" Great magazine & concept."

-Lee Gilchrist -firstrepublic.com

"Read past months - good content, plenty to read/review. Thx S"

- Scott Green -playup.com

"Nice magazine!"

- Huib Schoots

"Your magazine is a pleasure to read.

- an agile software developer .

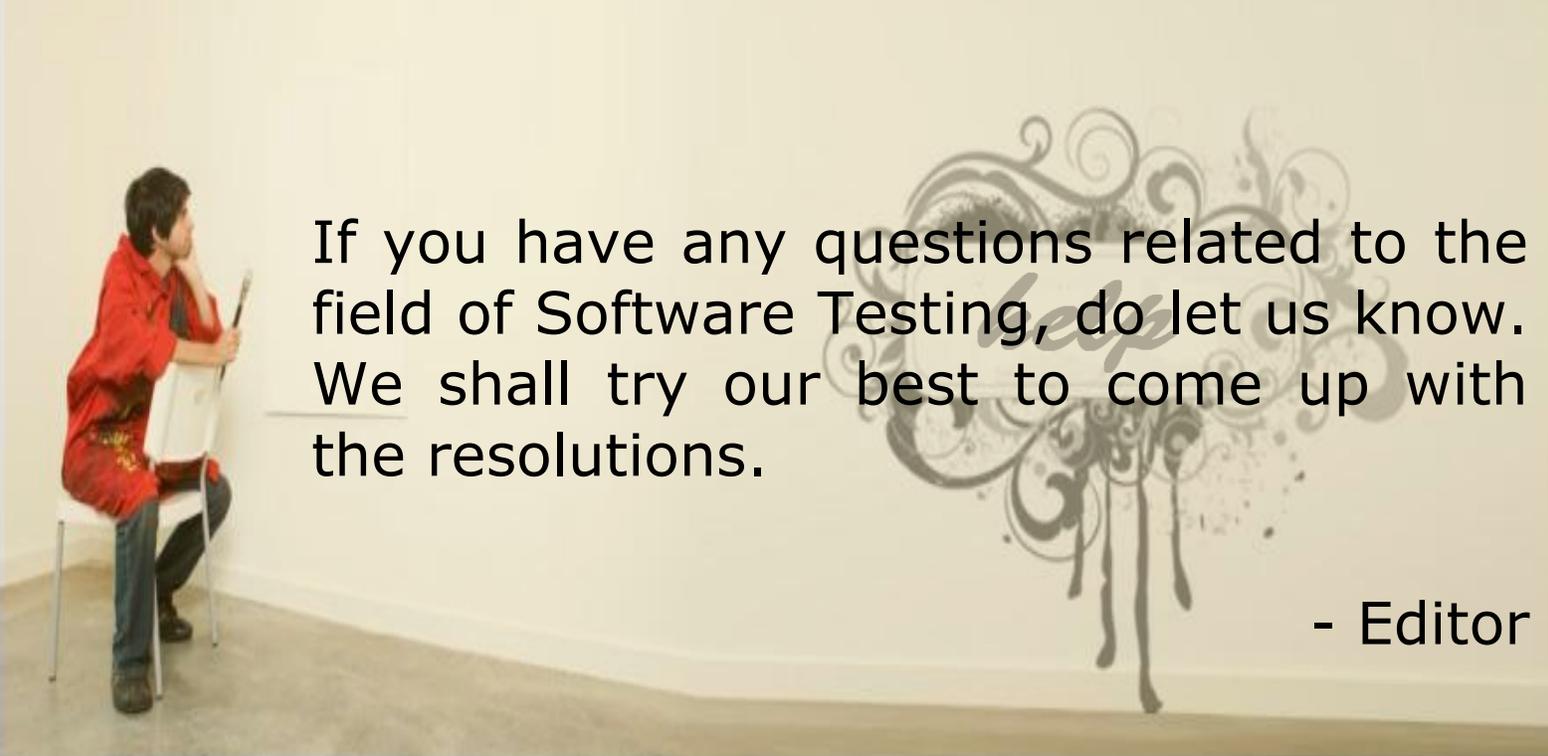
- Wim van de Goor

"I appreciate your efforts in promoting the software testing community."

- Ramkumar - Stagsoftware.com

You ask...

We'll help...!



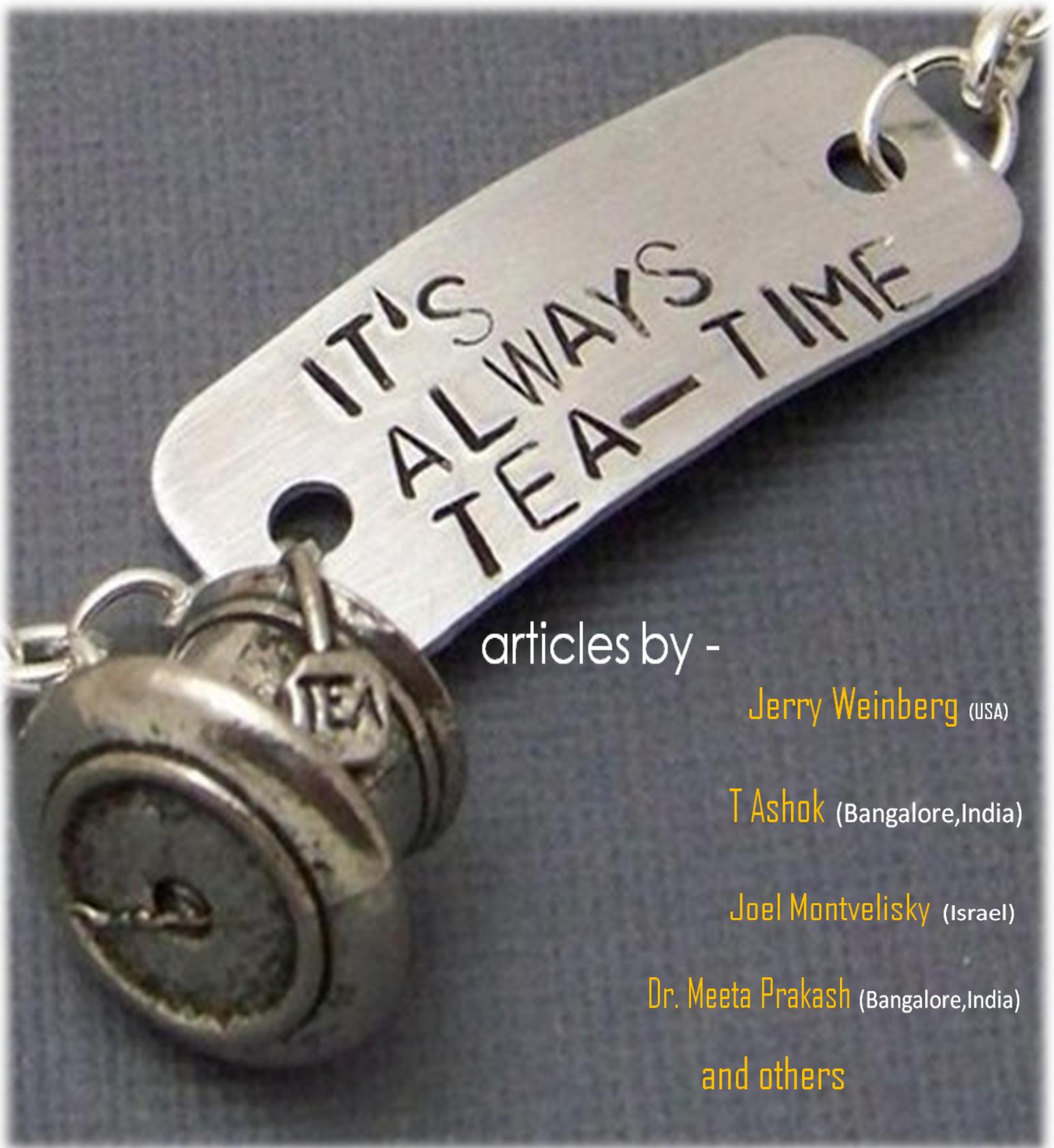
If you have any questions related to the field of Software Testing, do let us know. We shall try our best to come up with the resolutions.

- Editor

Feel free to write us your expectations.
Help us to help you better.

We are just a mail away: teatimewithtesters@gmail.com

in ne>xt issue



IT'S
ALWAYS
TEA-TIME

articles by -

Jerry Weinberg (USA)

T Ashok (Bangalore, India)

Joel Montvelisky (Israel)

Dr. Meeta Prakash (Bangalore, India)

and others

our family

Founder & Editor:

Lalitkumar Bhamare (Mumbai, India)

Pratikkumar Patel (Mumbai, India)



Lalitkumar



Pratikkumar

Editorial | Magazine Design | Logo Design | Web Design:

Lalitkumar Bhamare

Cover Page – Master Merchants International

Core Team:

Kavitha Deepak (Bristol, United Kingdom)

Debjani Roy (Didcot, United Kingdom)



Kavitha



Debjani

Mascot Design & Online Collaboration:

Juhi Verma (Mumbai, India)

Romil Gupta (Pune, India)



Juhi



Romil

Tech -Team:

Subhodip Biswas (Mumbai, India)

Chris Philip (Mumbai, India)

Gautam Das (Mumbai, India)



Subhodip



Chris



Gautam

*// Karmanye vadhikaraste ma phaleshu kadachna |
Karmaphalehtur bhurma te sangostvakarmani //*

To get **FREE** copy ,
Subscribe to our group at

Google™

Join our community on

facebook.

Follow us on



www.teatimewithtesters.com

