THE INTERNATIONAL MONTHLY FOR NEXT GENERATION TESTERS

# Tea-time with Testers

FEB-MARCH 2013 | YEAR 3 ISSUE I

## Jerry Weinberg

**Intelligence, or Problem-Solving Ability**

## James Bach

**Rise of The Intellectual Software Tester**

## Johanna Rothman

**How Technical does a Tester have to be?**

## Leah Stockley

**Addressing the Risk of Exploratory Testing**

## Bernice Ruhland

**Using Mind Maps to Organize Your Cross-Browser Testing**

## T Ashok

**Reflect and Change**

## Joel Montvelisky

**Spring cleaning your testware**

## Over a Cup of Tea with Michael Bolton

There are more than 40 leading organisations that host Tea-time with Testers on their knowledge portal.

**WHAT ABOUT YOURS?**

TEA-TIME WITH TESTERS

**Subscribe here Right Away to get  our all Issues for FREE**

# TEA-TIME WITH TESTERS

**First Indian Testing Magazine to reach 101 Countries in the world !**

**Created and Published by:**

***Tea-time with Testers*.**
Hiranandani, Powai,
Mumbai -400076
Maharashtra, India.

**Editorial and Advertising Enquiries:**

Email: editor@teatimewithtesters.com
Pratik: (+91) 9819013139
Lalit: (+91) 8275562299

This ezine is edited, designed and published by ***Tea-time with Testers*.**

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

# Editorial

Dear Readers,

First of all, I want to thank you for your patience. This delay was unavoidable as we were working on our brand new website. Yes, a new website to serve you yet better. Good thing is our 'Apple' users too can access it now (and we are sorry for stretching it this long).

Another good thing is 'Tea-time with Testers' has completed two years this February. On behalf of our entire team, I would like to thank each one of you for your tremendous love and support. I will always be thankful to our contributors and columnists for their wonderful articles.

Keeping in mind the lessons learned in last two years, we are coming up with some more ideas that I'm sure you all will like.

Till then, please enjoy this mega issue of your favorite magazine. There is a lot of good stuff here this time too.

Allow me take your leave for now.

Yours Sincerely,

Bhamare

- **Lalitkumar Bhamare**
editor@teatimewithtesters.com

# QuickLook

Testing Puzzles

by Sebi

mage: www.bigfoto.com

## Experience Report- AST India Test Forum

By – Garima Pandey (BTCI,Pune)

It was usual Monday morning at work and one colorful email flashed in my mailbox. Yes, it was 'January-2013 issue of **Tea-time with Testers'**. An e-mail that most of the testers eagerly wait for, every month. :-)

In no time I downloaded the issue and started running through the content. More than anything else, a big banner advert of **'AST India Test Forum'** caught my attention. I had little knowledge of AST but never knew that it will come to India some day and that too in Pune. Wow, I said to myself.

AST India Test Forum was to be held in Pune on 13 February 2013. Without any delay, I clicked on the link and registered for the event. I was all excited since it was my first time to attend such event dedicated to Software Testing community.

Finally the day came and we reached the auditorium. There was tremendous positivity in that big room, as if every individual was happy to be associated with such a grand event. There were few faces like mine, all curious to know what's in store. Auditorium was over-flooded with more than 200 participants (compelling the late comers to stand behind :-)).

## And then the show started:

We all were greeted by Keith Klan, a member of the Board of Directors for the AST and Global Head of GTC Barclays who started off with AST overview. Keith talked about purpose of AST, its history, various initiatives in US and mainly about the mission of this wonderful group.

ST i.e. Association for Software Testing is a professional non-profit association that is dedicated to advancing software testing and strives to build a testing community that views the role of testing as skilled, relevant, and essential to the production of faster, better, and less expensive software products.

Any individual can became a member by paying a nominal fee and better you check out the offerings on AST website here.

the event was houseful

Keith then talked about the great talent that India hosts and the reason why the meet up was scheduled here. He talked about CAST conference which is hosted in US every year and that the plans of AST to expand it beyond USA so that other testers from different corners can also take part.

I must mention that Keith is fantastic speaker and all eyes were glued to him while he talked.

The crowd was overwhelmed by the session but this was not all. There was something more in store for the attendees, a panel discussion with some of the elite people in the field of Testing.

Justin Hunter : CEO – Hexawise,

Pradeep Soundararajan : Director – Moolya

Smita Mishra : CEO - QAZone Infosystems and

Keith Klain himself (Global Head of Barclays GTC and AST Board Member)

The very first question to the panel was what we all have been thinking about at every interview that we face. "What are the SKILLS that are expected out of testers?"

Pradeep started with stating courage as the most important quality that he looks for in candidate. He very well justified that people should have courage to speak the truth to their stakeholders regardless of how bitter it may be.

Keith mentioned curiosity as one of the important skills that testers should have. Smita added the skill of conviction and other panel members too gave their valuable inputs.

*Keith Klain speaking about AST*

There were many other questions that were discussed like tools vs. skills, why senior management is expected to have hands on practices, various maturity levels associated with testing designations. Justin Hunter explained how tools can assist testers and how they should be used effectively.

**Certain things that I learned from this discussion:**

Testing is a team sport and not individual's game. A good test team comprises of good manual testers as well as testers that are good with tools and have programming skills.

On question on 'Skills and Compensation', what Keith answered impressed the audience. He said that every individual should know his market value. If he is excellent then company will offer him what he deserves. But he advised us to focus more on passion for our craft and continuous learning instead. And assured that good things will come to us on their own.

Needless to mention that Keith himself is great example of what he advised and we sincerely appreciate his efforts behind making this event happen.

Kshitij Sathe, who is Test Manager at Barclays did fantastic job by coordinating the panel discussion and made it even more interesting by adding some interesting questions from his side.

**Take Away:**

For a first timer like me, it was a treat to be a part of such a global event. There were so many questions that were asked in the forum, which we all might have had at some point of time and getting them addressed by such speakers was a dream coming true.

Special thanks to Keith, AST and Cognizant for organizing this event at Pune. And of course to 'Tea-time with Testers' for spreading the awareness and giving me opportunity to share my experience.

## Intelligence, or Problem-Solving Ability (Part 4)

Not very far—even closer than I thought. After writing the above package, I ran across the following ad in a trade magazine:

**TEST THE SKILLS OF YOUR PROGRAMMERS**

A program called XYZ consists of procedures, software, instructions and problems designed to evaluate programmer skills.XYZ measures the ability to write correct, compact, and efficient programs, and can be used to test experienced programmers or to evaluate the programming aptitude of non-programmers. The tests used are language independent, and are weighted in accordance with the difficulty of the problems and the time used to solve them. The XYZ program will run on a RST System. It is priced at $5,000 ..."

An evaluation of this ad is left as an exercise for the reader, but aside from its claim to test experienced or inexperienced programmers alike, it does offer one clue as to the proper direction for testing when it speaks of the "ability to write correct, compact, and efficient programs." What does this phrase suggest?

As one author comments, if we were hiring an oboe player for an orchestra, we would be able to give an audition, which would determine in a matter of a few minutes what his qualifications were, at least insofar as rejecting an obvious misfit. After

a longer performance, we would have a moderately good measure of the oboist's abilities. He then goes on to lament, "There is no equivalent of an audition" for programming. But why isn't there? Certainly there is not for a person without experience, but neither is there much for a child who is applying for a traineeship in oboe playing.

For a programmer who claims to have experience, why not just have him sit down and write a small program or read and interpret a set of specifications, sketching out his approach to implementing them? Why not, indeed? Perhaps it is because we so rarely read programs that this action never occurs to us, or perhaps it is because when we intend to hire an "experienced" programmer, we deem ourselves lucky if even a single applicant shows up. Testing a single applicant is like testing your wife

—if she fails, what are you going to do about it?

## SUMMARY

Everybody involved in hiring programmers has an opinion about what qualities are essential to a good programmer— even though we know that programming is no one single activity. Mayer, as cited previously, believes that his "X-factor" is related to aptitude." He speaks wisely, with the caution of one heavily involved in psychological testing. People more involved in programming are more confident of their impressionistic view. Not all of them, however, are as explicit about their X-factors as E. W. Dijkstra, who said:

I am engaged In teaching, at graduate level, in producing one variety of "mathematical engineer." The most powerful test that I know of for an applicant to be one of my students is that he have an absolute mastery of his native tongue: you just need to listen to him.

Since Dijkstra is "engaged in teaching," we cannot be too swayed by his remarks, for he may indeed have isolated a factor in academic success, as have several other workers. "Verbal ability" is as good a measure as we have of academic success in programming, so Dijkstra's intuition is probably a good one. But for actual programming performance, on commercial programs rather than "toy" programs, we lack any aptitude measure at all, except perhaps for general intelligence. For myself, I believe that intelligence has less to do with the matter than personality, work habits, and training. These things, unlike intelligence, can be changed by experience later in life, which turns the problem from one of selecting programmers to creating them. In other words, good programmers are made, not born; therefore we should turn our attention to the manufacturing, or training process.

## QUESTIONS

### For Managers

1. Do you use any aptitude tests now in choosing programmers, or does your personnel department use them? If so, what do you know about their validity?

2. Do you make any effort to validate them by evaluating programmers after a period of time on the job? What methods do you use for this evaluation? How convinced are you of their effectiveness?

3. What single important aptitude do you find most often lacking in your programmers? What kind of test do you think would discover that such an aptitude was missing?

4. Would you spend $5000 for the package in the ad? Or $100 per test? If not, why not? If so, have you been reading this article?

### For Programmers

1. Were you tested for aptitude when you applied for your present job? If so, did you find the test relevant? Do you know if your organization still uses these tests? If they do, and if you feel they are not relevant, what are you doing about it?

2. Do you feel it is possible to test for at least certain crucial aspects of programming aptitude? Make a list of what you consider to be crucial aspects of programming aptitude, and a list of suggestions as to how they could be tested.

3. Has this chapter indicated to you any areas of problem-solving ability or habit in which you may be deficient? If so, what plan do you have for doing something about it?

## BIBLIOGRAPHY

Mayer, David B., and A. W. Stalnaker, Use of Psychological Tests in the Selection of Computer Personnel, SHARE/GUIDE Presentation, October, 1968.

Reinstedt, R. N., et a/., Computer Personnel Research Group Programmer Performance Prediction Study, The RAND Corporation (RM-4033-PR), Santa Monica, California, March 1964.

Biamonte, A. J., Predicting Success in Programmer Training, Proceedings at the Second Annual Computer Personnel Research Conference, Association for Computing Machinery, New York, 1964.
Gotterer, M., and A. W. Stalnaker, Predicting Programmer Performance Among Non-preselected Trainee Groups, Proceedings of the Second Annual Computer Personnel Research Conference, Association for Computing Machinery, New York, 1964.

Okimoto, G. H„ The Effectiveness of Comments: A Pilot Study, IBM SDD Technical Report TR 01.1347, July 27, 1970.

Luria, A. R., The Mind of a Mnemonist, New York, Avon Books, 1968.

A fascinating example of how an individual can use his peculiar abilities or disabilities to achieve fantastic performance-

— in this case on tasks involving memory.

Hunt, J. M., Intelligence and Experience, New York, Ronald, 1961.
Hunt reviews the process by which the intellect develops, with an end to seeing what can be done to influence intelligence. Unfortunately, it seems that by the time one is old enough to become a programmer, most of the opportunity for modification has been lost. Perhaps we should start younger. Now that more and more programming fathers and mothers have computers at home, can it be long before we produce a programming Mozart?

Wertheimer, M„ Productive Thinking, revised ed. New York, Harper, 1945.

Wertheimer is the main founder of the Gestalt school of psychology, and in this book gave his prescription—based upon his experiences and experiments with young children—for teaching people how to think more productively. It seems unlikely that many of the ideas can be applied directly to teaching programmers to program better, but the book certainly is rich in suggestions as to which things to try and which to avoid.

Polya, George, How to Solve It, Princeton, Princeton University Press, 1946.

Polya, one of the great mathematicians of our time, gives a popular version of his work on techniques of solving mathematical problems and puzzles. Again, the work is highly suggestive for programming, but nobody has put it to a direct test, though many programmers have read it.

Ghiselin, Brewster, The Creative Process, Berkeley, California, University of California Press, 1952.

People have been studying the "creative process" for a long time—since Plato and Aristotle, at least—and we are a long way from understanding it. We don't know, even, if programming really requires creativity, or creativity on the level of a Mozart or an Einstein, or even if there are different levels of creativity, or just different levels of creations. But for those who suspect that there may be some element of creativity in programming, this collection of original works and excerpts will certainly add fuel to their flame. Even for those who don't believe in creativity in programming, who could miss a chance to read what Einstein, Mozart, van Gogh, Wordsworth, Coleridge, Yeats, Nietzsche, Jung, and Spencer—to name but a sample— have to say about the subject. Besides, this is the source of the Stephen Spender misprint (in the paper edition).

Sackman, Harold, Man-Computer Problem Solving, Princeton, Auerbach, 1970.

Sackman's studies are not concerned solely with programming, but with the more general category of problem-solving behavior. His book contains many useful insights into the effects and lack of effects that working with a computer has on problem-solving behavior, but in a certain way he misses the point on programming as problem solving. Sackman himself is not a programmer, but a psychologist, and the one thing he never seems to have done in his studies was to read the programs produced. Like most psychologists, he apparently assumes, that all completed solutions to the same problem are equivalent, and that only failures to complete a program and get the correct results need be considered carefully. But all programs that give the same output are not alike, as we know, and until psychologists recognize this, most of programming/problem-solving behavior will remain a fog.

Weinberg, G. M., Experiments in Problem Solving (Doctoral Thesis), Ann Arbor, University Microfilms, 1965.

Recommended for the psychologist who does not understand the commentary in the previous citatio. This study shows how sufficient resolution in the observation of the problem-solving task and the ability to characterize the task in terms of the individual subject's strong and weak characteristics lead to striking insights into problem solving.

Such insights will never be available if the level of resolution remains too high, or if individuals are averaged together to get "statistically significant results."

Back To Index

## Comments on: Intelligence, or Problem-Solving Ability

This article series has an interesting "now-it-can-be-told" history. Over the years, people have asked why I devoted so much space to programmer aptitude testing, only to conclude that it was all nonsense. The truth is the original manuscript discreetly ignored this subject but two reviewers of the manuscript happened to be in the business of selling such tests. Naturally, they insisted that the book address "the most crucial subject in computer programming."

To satisfy my original publisher, I wrote the parts of this chapter addressing aptitude testing.

Today, programmer aptitude testing is a dead issue (or so I hope). If this chapter had any part in killing this ridiculous practice, then I take pride in having done my job. Yes, intelligence is a factor in programming success— people with an IQ less than 50 are probably not going to create operating systems (although they might design new languages—the Devil made me say that).

There are many factors besides intelligence that contribute to programming success (or failure). I've known some rather dull people who consistently turn out fine, useful code and some rather brilliant people who never turn out any useful code at all. In short, I'm sticking with my twenty-five-year-old position: "... intelligence has less to do with the matter than personality, work habits, and training. These things, unlike intelligence, can be changed by experience later in life, which turns the problem from one of selecting programmers to creating them."

By the words "creating them," I don't mean forcing them to follow some "best" style of thinking. My own doctoral thesis showed quite clearly that problem solving style was unique to the individual. Thus, anything that forces a programmer to think in another person's style reduces that programmer's ability to solve problems. The greatest challenge, then, is not creative thinking, but creative communicating: representing our thoughts in a way that other persons—each with a unique style—can understand.

# Biography

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

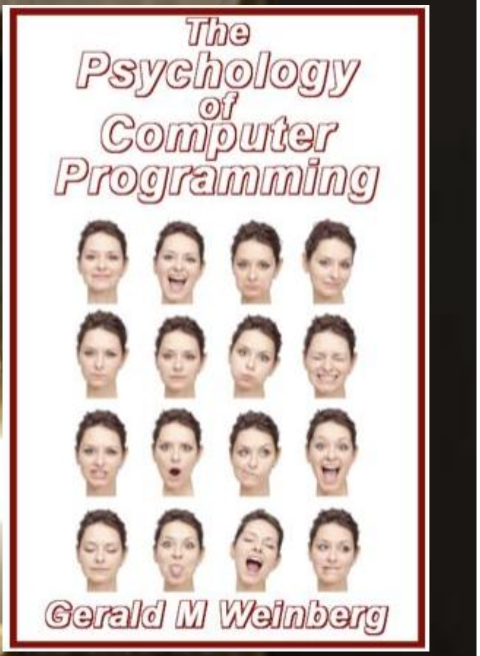
For more than 50 years, he has worked on transforming software organizations. He is author or co-author of many articles and books, including The Psychology of Computer Programming. His books cover all phases of the software life-cycle. They include Exploring Requirements, Rethinking Systems Analysis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

Jerry's another book **The Psychology of Computer Programming** is known as the first major book to address programming as an individual and team effort.



**TTWT Rating:** ★★★★★

"Whether you're part of the generation of the 1960's and 1970's, or part of the current generation . . . you owe it to yourself to pick up a copy of this wonderful book." says Ed Yourdon, Cutter IT E-Mail Advisor

Sample of this book can be read online here.

To know more about Jerry's writing on software please click here .

Speaking Tester's Mind
- straight from the author's desk

# The Rise of the Intellectual Indian Software Tester

## Part 3

- by James Bach

[Please read part 1 and 2 of this series in previous issues of Tea-time with Testers]

India. Yeah! But first, I need to talk about training testers, because that's a special problem in India.

Training is sometimes assumed to be a process of transferring knowledge—as one might deliver a package from a truck. Actually, that model does work for simple things. If I "teach" you my phone number, I'm essentially dropping it at your front doorstep. But for complicated, subtle, or otherwise difficult to practice skills, it can cause a lot of harm to think of teaching that way. A student who has any trouble learning, and yet believes learning is just a matter of "taking in" a delivery of "knowledge" will soon begin to feel stupid, and stop even trying to learn.

The trouble is, as a teacher, I can't actually open your mind and pour the knowledge in. Your mind can't open that much and the knowledge is too big. So, at the very least, teaching is like building a ship in a bottle, piece by piece.

No, it's not like that either, because a bottle is just a space surrounded by a barrier. Your mind is a different kind of thing. There's no "empty space" in it. Every space is already taken. To learn is to create a new space and connect it to all the other stuff in all the other spaces. It's a construction process, and no teacher can do that, only the student can. Therefore, all teaching is a process of helping the student's own learning process. All learning is self-learning with various degrees of assistance and stimulation from the outside. It's like doing surgery on yourself, while the surgeon tells you what to do.

And, no, I still haven't got it, because no one can tell you how exactly to construct yourself. And there's more. Learning is also a process of adjusting, or unlearning, lots of outdated or broken old ideas. Learning is also a process of changing your identity; slowly coming to feel like an expert. It's also a process confronting, challenging, and sometimes accepting your own limitations. Learning is personal and social; tacit and explicit. It is mental and physical and emotional. Whew!

So, a teacher helps… somehow… in an indirect way. But learning is mostly about the learner and under the learner's control, whether he realizes that or not.

## My Teaching Method

I find it useful to a baking analogy to describe my particular indirect method of teaching. A baker applies energy to create chemical changes, otherwise the bread doesn't happen. The energy I work with is personal enthusiasm, nurtured through experiences of enjoyable success in testing.

1. **Attract people who might want to learn testing, and filter out the people who probably won't learn from me. ("Start with the right ingredients")**

a. Be excited about what I do and public about how I do it, so that right students find their way to me, and the right ones stay away.
b. Make them think I care about them by using the tactic of actually caring about them.
c. Have high *aspirations* for each student, but keep *expectations* low at first.
d. Always tell the truth about how they are doing, but work a little harder to find hopeful news than to find discouraging news.

2. **Acknowledge and begin to disarm their secret fears ("get rid of any roaches in the kitchen"):**

a. That they aren't smart enough to test.
b. That testing is a waste of what talents they do have.
c. That testers are unlovable.

3. **Get them hooked on testing excellence ("start the fire") using the following tactics:**

a. Focus first on the thrill of discovery.
b. Give them an experience of the joy and power of their own curiosity.
c. Demonstrate the connection of testing to something they already know, so that they don't feel like hopeless beginners.
d. Provoke them to test intuitively, watch for moments of talent, then reveal evidence of their talent to make it tangible for them.
e. Give them a sense of what mysteries are revealed to those who study testing deeply.

4. **Challenge their naïve beliefs about testing ("clean the kitchen") using the following tactics:**

a. Bring their mental models of testing to the surface through conversation and practical exercises.
b. Honor the good intent and any grains of truth to be found there.
c. Use practical exercises, Socratic coaching, or direct explanation (in order of preference) to reveal weaknesses in typical folk beliefs about testing.
d. Demonstrate and explain powerful alternatives.

5. **Give them permission to reinvent testing for themselves, ("let them cook") and invite them into the community of ethical, creative software testers.**

a. Get the student to try to solve a problem before telling him how to solve it.
b. Teach the identification, development, and proper use of heuristic methods.
c. Expressly invite the student to create modified versions of models and tools.
d. Demonstrate the process of context-driven process engineering.
e. Remind them "if you only know one way to do something, you are a robot, not a thinker."
f. Discuss ethical challenges and share stories about way to meet those challenges.
g. Celebrate heroes and acts of heroism. Offer credit where credit is earned.
h. Interact on public forums. Regularly refer to people we respect.
i. As a teacher, discuss my own struggles and ethical regrets.

6. **Challenge them to practice the difficult aspects of testing, under various kinds of pressure ("bake the bread"):**

a. Hold peer conferences to examine and compare experiences.
b. Provide and receive coaching.
c. Practice with documentation.
d. Practice with tools.
e. Practice with mathematics.
f. Practice with written and oral reports.
g. Practice under time pressure.
h. Practice under critical attention of peers.
i. Practice in public.
j. Practice with unfair project conditions.
k. Practice dealing with failure.
l. Practice in unfamiliar contexts.
m. Write about these experiences.

7. **Acknowledge and celebrate successes, improvements, and boldly achieved failure ("enjoy the feast and thank the cook").**

## What About India?

I was getting to that… My point is that having good idea, even the "best" idea, is a small part of getting other people to have good ideas, too. Just knowing how to test does not in any way make us good teachers of testing. No wonder that in all the years software testing has been a commonplace and accepted part of software development, it has barely advanced as a recognized craft. No wonder the ISTQB can package outdated and demonstrably wrong testing folklore into a syllabus and successfully make people pay to get "certified" in it. Testing is hard to teach, it's hard to observe, and that makes us easy targets for fakers and quacks who want to make a buck.

Most of the testing world remains, to this day, in a dark age. But there is hope. The testing industry cannot be changed from the outside. And no one person can force it to be better, even from the inside. But small groups of testers who understand community dynamics and *who work on themselves first*, can drive a process that eventually reinvents the testing craft. This is already happening.

I believe if the Indian testing industry is to become vastly more productive than it is now—and more respected—its practitioners and clients must profoundly change how they think and talk about testing. They must leave the factory and manual labor metaphors behind. They must embrace testing as a process of skilled, creative design. They must come to value effective critical thinking, not key presses per second or test cases executed per hour. They must refocus their process improvement efforts on three things: developing skills (rather than procedures), developing heuristics (rather than standards), and developing supportive tools (rather than tools that alienate us from our own testing process).

That's why I described my teaching method, above: my theory is that the same basic approach works whether you are developing yourself, or helping *whole countries of testers* to develop.

**Notice step one of my method**. It's not about explaining anything. It's social. It's about attracting the right people and repelling others. Yes, the craft can be improved through a direct assault on bad ideas and the people who hold them. You can stand on a chair and shout, but most people won't listen to you in the right way unless they feel you are part of their community. Social and personal factors dominate rational factors in humans every time.

So the rise of the intellectual Indian tester is going to happen mostly along the line of social connections. It begins with a small community, but they will attract more, by virtue of their reputation for excellence and the inherent pleasure of testing with our minds.

**Notice step two.** We have to deal with the fears of the testers who may want to adopt this new way of thinking about testing. Partly, there is a perception of safety in numbers— and that's what will keep most testers in the other camp of factory-style testing—but over time this will shift. Meanwhile, we need to offer what support we can on the forums, in conferences, and peer to peer. Just tonight I was Skyping with yet another Indian tester who needed a pep talk for how to deal with a bad manager. We must make it part of our job to help each other that way.

**For step three** we need public outreach in the form of testing parties and open forums. That's the fun part of growing community.

**Step four can be a bit messy.** Challenging bad beliefs is best done within the community of committed intellectual testers, but it can be difficult to arrange that in any sort of public forum. An excellent way to do it is to hold peer conferences—which are small gatherings of invited testers who come together to share experience reports and then to face hard questioning from their peers. Again that's going to work on a small scale, and for people who are already committed enough to come to a peer conference. What about the rest?

Some of my colleagues suggest avoiding public debates. I'm famous for starting arguments on Twitter with people I feel are bad testers. It think it's important to do some of that, at least, even though debating is often not much fun. It's important because otherwise new testers will think that the Factory Schoolers own the craft. I understand why a lot of my colleagues want to be nice to testers and consultants who push ISTQB certification, "100% automation" and other silly ideas. They think being nice will change the prevailing practices in the world. On the other hand, I've seen this play out over the last 25 years and it seems to me that being nice hasn't worked *even a little tiny bit*. Working with the ISTQB people did not stop them, it helped them.

I will say this: if someone is sincerely trying to learn testing, be friendly to him. Otherwise, don't waste your time hoping that charm will win the day. In that case debate is the better path. Let's shake things up. Let's get louder.

The rest of the steps also relate to the transformation of Indian testing, but I'll leave them as an exercise for the reader, at least for now. It's time to talk about the last part of my India trip.

## My Visit to Barclays

The same dynamics work for developing a company culture of skilled testing, too. Companies must hire carefully and have some method for ridding itself of people who aren't getting the job done; they must drive out fear; they must engage workers minds, and not just assume that paying them creates all the motivation required. All these things map to the steps I outlined.

I've been working with Barclays Bank for a couple of years now. They are going through those steps, more or less, as a corporate testing culture. It's a big organization, and big organizations can't change very quickly. They have a substantial number of testers working in India, now, which is why I came to Pune.

When I first worked with Barclays in Singapore, they selected a relatively small group of very ambitious testers for me to work with ("the right ingredients"). I began their training, and then they took the initiative to teach a version of my material to many others after I left. They created that special version of Rapid Software Testing training by themselves, modified from my materials ("let them cook"). They established several internal forums for discussing and encouraging the learning ("starting the fire and cleaning the kitchen"). Over time, some of those testers have come a long way ("baking the bread"), and one systematic approach they used is a testing competition spread over a year.

Changing an existing organization of 700 testers while they are working on projects everyday is a slow process. They've been working at it a couple of years now, though, and it's coming along well. But there's a new twist: Barclays now has a large contingent of Indian testers, and Keith Klain, the leader of their Global Test Center, wants to create the same skilled testing culture there as in Singapore and his other sites.

I thought that might be a little challenging. It really wasn't. Teaching my class at their Pune site, I was once again impressed with the high quality of the students. They were young, ambitious people who dived into the material with enthusiasm. The right ingredients.

Afterward I judged the last part of the 2012 "Super Tester" competition. The task for the finalists was to test XMind using tools. We didn't specify how the tools should be used, only that the testing would be judged on the basis of effectiveness and originality. This is an example of "baking the bread"—testing under time pressure and scrutiny. Here the result was a little disappointing: because none of the testers did much with tools. I had suspected that the testers may not be comfortable with tools, and that's why I wanted to give them this challenge. Now they know they need to practice harder to become more comfortable with the use of tools in testing. This is why it's important to push each other. We must discover our weaknesses so that we can deliberately work on them. A testing competition is a relatively safe place to do that.

## The Challenge For India

What is an intellectual software testing culture? It's people who feel comfortable and eager to face any complex technical and philosophical problem that comes up in their projects. It's people who understand that excellent testing is difficult to do, and who take responsibility to educate not only themselves, but also their clients and colleagues about how it can be done. It's a self-perpetuating social order of skilled testers who are valued and supported by the intellectual software developers with whom they work.

I visited three other large companies in India, before I left. I saw a lot of what I expected, and some things I didn't. I didn't expect much enthusiasm for exploratory testing, for instance, but instead there was quite a lot. Everybody I talked to wanted to know more about it. The spark of an intellectual testing culture is there.

What surprised me a little are the questions I was asked about exploratory testing: How do you measure it? How do you document it? How do you estimate it? I was asked these questions as if "normal" testing is easy to measure, document, and estimate. Of course it isn't. Of course the fact that you think about and develop testing as you go rather than write it out beforehand does not in any way make it harder to measure, document, or estimate. Testing is always hard to measure, etc. Scripted testing *seems* easier when it comes to those things, because scripts are visible. Except that scripts aren't testing.

These questions come mostly from management. That's the challenge, then. Management in India has to get serious about testing. They need to learn how to test, and break out of their illusions about counting little test cases as if they were units of testing goodness when as I just said, scripts aren't testing. As I wrote in part two, Moolya has figured this out. Parts of Barclays have figured this out. But management in the rest of India seems to have a long way to go.

Back To Index

**James Marcus Bach** is a software tester, author, trainer and consultant. He is a proponent of Exploratory testing and the Context-Driven School of software testing, and is credited with developing Session-based testing.

His book "**Lessons Learned in Software Testing**" has been cited over 130 times according to Google Scholar, and several of his articles have been cited dozens of times including his work on heuristics for testing and on the Capability Maturity Model. He wrote numerous articles for IEEE Computer.

Since 1999, he works as independent consultant out of Eastsound, Washington.

He is an advisor to the Lifeboat Foundation as a computing expert.

Follow James on Twitter @jamesmarcusbach or know more about his work on satisfice.com

# Addressing the RISK of Exploratory Testing part 1

- by Leah Stockley

In my experience, by far the biggest risk to successful implementation of Exploratory Testing is the complete misunderstanding of what is meant by the term.

Three years ago I was the first test manager in my company (to my knowledge) to formally introduce Exploratory Testing (ET) as a phase in a testing project. I easily articulated some of the benefits it would bring the project. Earlier testing = earlier clarification of requirements, not to mention earlier, and therefore cheaper defect detection. I was able to deliver these benefits to the project, but my lack of true comprehension with regards to the definition of ET meant I did not fully understand how to support my team, who were also trying it for the first time.

In this series of articles I will share the concerns that have arisen whilst implementing ET and some potential solutions to overcome them. This first article will address:

## How to gain the confidence of the test team

After the first attempt at implementing ET, I attended James Bach's Rapid Software Testing training course. I expected 5 days training on Exploratory Testing. What I actually gained was so much more… 5 days of training on RST, of which ET is just a small, but essential part. James' course made me realise the wealth of underlying skills and techniques that anyone calling themselves a 'tester' should strive to develop. It was these skills and understanding that my test team needed to develop, in order to be confident exploratory testers (not to mention better testers overall).

## Isn't Exploratory Testing just another name for Ad-hoc Testing?

This is often the first misconception that arises during the training courses I run. Of course it could mean ad-hoc… in order to run ad-hoc tests you would most likely be exploratory testing. However we can still gain the benefits of ET (early testing, early effect detection, less script maintenance and rework), whilst placing sufficient structure around it to be confident of the coverage we will achieve.

More and more I use the term 'Structured Exploratory Testing'. ET can be structured because:

- We are not just banging a keyboard without thinking about our actions or the result
- We do plan & communicate in advance what we are going to test
- We do analyse and understand what we expect to see when we test

But it is still exploratory testing because

- We have not planned the exact steps we will take to perform a specific test.
- The tester has freedom to choose the next best test to perform based on what they learned from their analysis, plus what they learn about the system whilst testing
- The test is repeatable but will not be performed in exactly the same way each time it is run

## How can a test be repeatable, if it cannot be run exactly the same way each time?

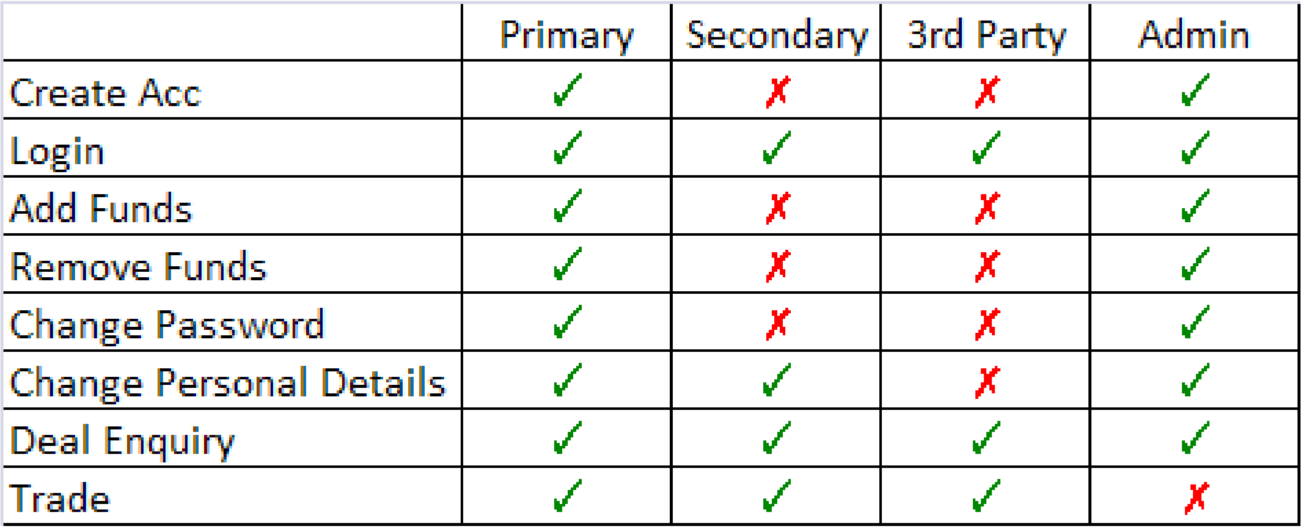
This is another concern often raised by testers. Some in the test industry have become so ingrained that a test must be performed exactly the same way each time (perhaps by companies selling very expensive automation tools, or very 'robotic' testers?) that they miss the value in running the same test in a slightly different way. Think… when you use MS word, do you use it exactly the same way each time? Sometimes I use the mouse or sometimes a keyboard shortcut to achieve the same action. If one user performs actions differently, imagine the variance when a system has multiple users! Perhaps randomness in test data is important… or the number of times you click on a button, as a user may do in frustration when the application appears to be frozen (which often causes a complete system crash.. try it!). Any of these variances could give you important, new information about the product you are testing… but would it be possible, valuable or ethical to spend your company time writing a detailed test case for each of these scenarios?

When we repeatedly run the same test, we simply confirm what we already know. Would it add more value to learn something new about your application? As long as you test the core functionality, it is often cheap, yet informative, to change the specifics of how you test. And if it really is critical that the test is performed precisely… then congratulations, you have identified one of those rare occasions where Exploratory Testing would not be suitable… so write a script!

So, let me define Exploratory Testing! When using the term 'ET' I mean **'simultaneous (and cyclical) learning, test design and test execution'**. With this definition in mind…

## How can testers be sure not to miss anything important when testing, if no scripts are written beforehand?

Firstly, I want to clarify that ET does not mean 'No documentation'. We created documents known as 'charters' or high-level test scenarios. We backed these up with lightweight but informative test matrices (See figure 1). As a manager these were sufficient assurance that my team was achieving the coverage required during testing. I had spoken with them and trusted they understood what and how to test, without writing it down first.



| | Primary | Secondary | 3rd Party | Admin |
|---|---|---|---|---|
| Create Acc | ✓ | ✗ | ✗ | ✓ |
| Login | ✓ | ✓ | ✓ | ✓ |
| Add Funds | ✓ | ✗ | ✗ | ✓ |
| Remove Funds | ✓ | ✗ | ✗ | ✓ |
| Change Password | ✓ | ✗ | ✗ | ✓ |
| Change Personal Details | ✓ | ✓ | ✗ | ✓ |
| Deal Enquiry | ✓ | ✓ | ✓ | ✓ |
| Trade | ✓ | ✓ | ✓ | ✗ |

Figure 1. Sample Test Matrix

What surprised me was that some of my team struggled with the absence of the test script creation phase. Personally, I found writing test scripts a waste of time, as the steps are rarely followed to the letter during execution anyway. I have since realised that for a lot of testers, writing scripts is the method they use to develop their understanding of the system, and therefore design their tests and the expected results. These testers were not comfortable letting go of a process they'd used for years, without having a new one to replace it. This removal of structure is a big cause of worry in some testers. They fear they may miss testing something important if they haven't written down their test ideas or expected results in detail first.

### Analysis is the key…

I have since understood that time not spent writing detailed test scripts is better spent actively learning about the system. In other words, performing test analysis. Learning about the risks, constraints and potential test coverage in order to develop a more informed and targeted test approach.

### …Analysis and Communication!

But the critical factor that ensures success… is communication!! I now enable my testers to be confident they will test the right things by giving them the time to talk with others and develop their test ideas. Sounds like it takes too much time? Without exception, teams doing this, in a structured way, have more quickly analysed the system and defined a valid test approach faster and more effectively than the teams who sit in silo's churning out test scripts.

We use a combination of brainstorming with Heuristic Checklists (google them!) and mind mapping to capture & present the results. This detailed analysis session (also known as a survey or intake session) provides clarity on the understanding of the system, and the ability to ratify that with other people. Once the system and risks are better understood, we can determine the coverage. The test design & execution approach is then discussed. And all this before anything is documented (although the documentation by way of test matrix or mind-maps is often created at the same time).

### Proactive teamwork, not rework

This replaces the traditional Peer Review with a more proactive and motivating process which encourages continuous learning and removes the need for rework. This has proven more efficient than designing tests on a solitary basis and having those design ideas (scripts) reviewed afterwards. Some projects include their Development or BA's in these sessions. Where this is not possible, all of them walk the Dev/ BA through the results of the brainstorming to get their opinion on the risks and test coverage. This improves relationships on the project by building trust, respect and knowledge throughout the project team. It's important to include your whole test team in this. What better way for your more junior team members to gain understanding of the system?

Having gained this clarity of understanding about what ET means, plus had time to analyse the system instead of focusing on documenting it, our testers emerge confident they know what is important to test. Confident because they have been allowed to openly discuss and resolve their concerns and knowledge gaps. They have confidence to proceed without creating detailed test scripts which, without fail, require future re-work and maintenance as the system inevitably changes.

In the next article, I'll share how we gain confidence from the project teams when adopting exploratory testing.

Leah has more than 14 years experience in Software Testing for consultancies, banks and software houses across multiple industries.

Her Context-Driven awakening began 2 years ago. In that time she has become passionate about inspiring and coaching testers to be innovative, empowered and to continually improve the art of testing.

Currently working at a large global bank, Leah is responsible for rolling out the CDT approach across the Global Test Centre and developing solutions to the concerns that arise when introducing & applying CDT.

When not at work, Leah shares her experiences on www.inspiredtester.com

Back To Index

Come in We're
HIRING

*There was a time when people did not have compass to find right direction. The only guide they had was that guiding star up in the sky.*

*Do you think that you are also stuck somewhere with technical issues? Do you need help in decision making or want guidance?*

*Well, the wait is now over . Introducing….*

# "The Guiding Star"

*The panel of our experts is now here to help you.*
*Send us your questions around software testing and our Guiding Stars will help you out.*

**E-mail your question on –**

**theguidingstar@teatimewithtesters.com**

Please Note :

1. This is not a job portal.
2. Typical interview questions will not be answered.
3. Questions should be on Software Testing or related topics only.

Image courtesy : MMVI New Line Production

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

No Problemo! We will publish your Feedback/Comments and also the answers to your Questions that you have for our Authors.

Do write us your Feedback and Questions in below format and send it to teatimewithtesters@gmail.com :

- Your Name
- Your Brief Introduction
- Article Name
- Your Feedback or Questions if any

Make sure to write **Feedback For < Article Name>** in your subject line.

In the school of Testing
for your better learning & sharing experience

# Hiring Great Testers [A How-to Guide]

by Johanna Rothman

## How Technical Does a Tester Have to Be?

After you've defined the kind of person you need, which you did in part 1 of this series, it's time to address how technical a tester needs to be.

If you look at the job ads these days, you can see a huge difference in what employers were looking for, from just several years ago.

Several years ago, organizations downsized their testing departments, thinking there was no value in testing. Or, they hired low-cost people to do perform primarily manual repeatable testing. The big question for years in our industry was this: What's the "correct" ratio of developers to testers?

Well, if you do what we have called "monkey" testing—that is, you don't think about the design of the test cases, and you don't explore the product, you don't get much value from the testing. You need a lot of testers for that kind of testing. And, you have second-class testers and you need a lot of testers.

But that kind of testing has never served our industry well.

## Agile Has Changed Everything

So, what's changed?

Agile has changed everyone's expectations of what we can expect of our testers. We always needed to expect this from our testers, and I'm delighted we can expect it now. Now people need testers who are technical. But, how technical does a tester have to be?

First, let's discuss what technical expertise is.

There are four areas of technical expertise: functional skill, domain expertise, tools and technology, and industry expertise. When you ask how technical a tester has to be, you are asking four separate questions.

## What functional skills do you need?

You might need testers who can write automated tests. In that case, your testers are going to look something like developers. But, how much design skill will they need? Will they need to know how software is designed, or will they need to know how to design software? There is a big difference between those two skills.

For example, I understand different system architectures, such as distributed systems and multithreaded systems. I understand how to test them differently and where they are likely to fail. But don't ask me to design them. No, sirree. I would not be good at that. Why? Because I have built my functional skills in testing those systems, not in designing them. I understand many different kinds of architectures. I have studied architectures. I can read and write code. But that is different from being the main person who can drive the architecture or design of a particular kind of system. There are testers who are capable of being system-level architects.

That's just one potential functional skill. You might not need that skill. But on an agile team, *someone* needs the skill to develop automated tests.

Someone needs the skill to create representative scenarios for the normal case, abnormal case, performance, and reliability. You might think the product owner or the business analyst might have a hand in doing this, but my experience says the tester must contribute here. The product owner might be able to define the normal case of acceptance criteria for a story, but is likely to forget any edge cases. And, forget about abnormal cases, performance scenarios, or reliability testing. This is where the tester shines.

These skills are not limited to agile testers—not at all. But on agile teams, this is where I see teams most often say, "Who will do this test definition? Who will convert this conversation to an automated test?"

That's when you need testers who know about equivalence partitioning and combination testing techniques. You don't want people who run the same test 55 times, because they don't realize it's the same test. You want people who run 55 unique tests.

## What functional skills can you train?

Maybe you can train testers to test in exploratory techniques if they don't already know. I find that testers who don't yet know session-based test techniques are quick to learn them.

Exploratory testing is not monkey-testing. And, for some people, it's difficult to learn. With proper training, people can learn exploratory testing. You need to be able to experiment and have a sense of curiosity: what will I learn from this test? Where can I break this product? What will this test teach me about this system? Where can I go from here?

If you are accustomed to planning the tests in advance and having a set plan, exploratory testing can be scary. Especially combined with test coverage, it's not trivial. Yet, you want to invest in the functional skills of exploratory testing and test automation. It sounds as if it's an oxymoron, but it's not. These test skills come at the product from different directions, so you want people who can do both.

People can learn test coverage and learn what they want to cover. What makes sense to cover? What does not make sense? You want people who can look at the tests and the product and who have the judgment to say, "We have tested enough for now." Those are people who can weigh risks and are articulate enough to discuss those risks with you.

Do you see now, why it's so important to discuss the qualities, preferences, and non-technical skills before you start discussing technical skills? (you can read my first article here)

## What domain expertise do you require?

Domain expertise comes in two flavors: problem-space and solution-space. Problem-space domain expertise is when the tester understands the product's problem. Solution-space domain expertise is when the tester understands the internals of the product, the product's solution.

I expect every tester to learn enough about the requirements to gain expertise in the problem-space. Testers need to be able to discuss user stories with product owners or developers and to discuss acceptance criteria with anyone. Acceptance criteria often turn into tests of some variety. If you are not agile, testers need to be able to discuss requirements with product managers, business analysts, developers, anyone on the project, and develop tests based on those requirements. Problem-space domain expertise may be complex, but is the minimum required for any tester.

The real question is this: how much expertise does your tester need in the solution-space? Some testers require expertise in the architecture of a system to be effective. Some don't. Some testers need to be able to test performance or reliability. Some don't. Your mileage will vary.

What does your product require? Are you testing a transaction processing system? I suspect you need to test performance. Are you testing a database? I would test reliability. Are you testing for a bank or some other application where people's money is at stake? I would test security, backwards, forwards and inside out. I would make it my business to understand the architecture of the system. I would make the developers tell me how the system works.

## Take a developer to lunch

The best way I know to gain solution-space domain expertise is to take a developer to lunch. Not at a fancy-schmancy place. If your company has a cafeteria, that's best. Sit down next to a developer and ask him or her, what's happening in the code. Then listen. Or, ask the developers to provide you and the rest of the testers a series of lunch-and-learns, where they present the internals of the system.

Whatever you do, make sure you gain solution-space domain expertise if you need it. And, many testers do need some of it.

## Tools and technology expertise

This is what many ads lead with, which is easiest to learn: test tools, operating systems, coverage tools. You've seen examples of these: Fitnesse, Selenium, TeamFoundation Server, tools like that.

Now, I don't know how important those tools are to your testing. I suspect they are pretty important. How long would it take for someone to learn the tools you have at work?

My experience with tools is that it doesn't take too long to learn a new tool. Humans are tool-users. We learn new tools all the time. When we rent cars, we learn how to use a new car. It's similar enough to the car we have at a home that we know enough to drive it. The first car we learned to drive? That was difficult to learn. The next car? Easy.

### Industry expertise

Industry expertise is where we have a lot of tacit and implicit knowledge, and that's dangerous. Once I assume you know something, all bets are off. You might. You might not.

The problem with implicit knowledge is that it does not always transcend geographical borders. What is common knowledge in one country or culture is not common in another. This is especially true given the state of regulated industries. Every government regulates them differently.

We have some common areas of industry expertise: in healthcare, I assume that testers realize that security is an issue because of HIPAA, the Health Insurance Portability and Accountability Act in the USA. Anywhere you have medical records, you have the issue of data, and anywhere you have data, you have the issue of security. This would also be an issue in banking, another regulated industry. The insurance industry has different state and country regulations.

Any regulated industry has its own language, TLAs (three letter acronyms), and its own concerns.

Again, once you start crossing borders, what people "know" to be true might be unknown to other people. Testers might know terminology and have a start on the right questions to ask. And, that would be a great start.

### How technical does your tester have to be?

I hope you decide that the best answer to this question is, "It depends." You need to analyze the open position, and decide that some technical skills are essential to the job. Some will be just desirable. Based on your culture, and the team, and how you bring the tester into the organization, you will need a more-or-less technical tester.

**Johanna Rothman** is the author of Hiring Geeks That Fit, https://leanpub.com/hiringgeeks.

See her other books at http://www.jrothman.com/books/.

She writes an email newsletter, the Pragmatic Manager,

http://www.jrothman.com/pragmaticmanager/

Back To Index

Taking
a break?

a click here
will take you there

# Using Mind Maps to Organize Your Cross-Browser Testing

By Bernice Niel Ruhland

During this cross-browser testing series, I shared information gathered from many testers on how they approach cross-browser testing. You can read those articles in the June, July, and August 2012 issues. My article in the October 2012 issue discussed balancing time with cross-browser testing. For the final part of this series, I am sharing sample mind maps that I find useful in organizing my testing.

### Create a Mind Map to Stay Organized

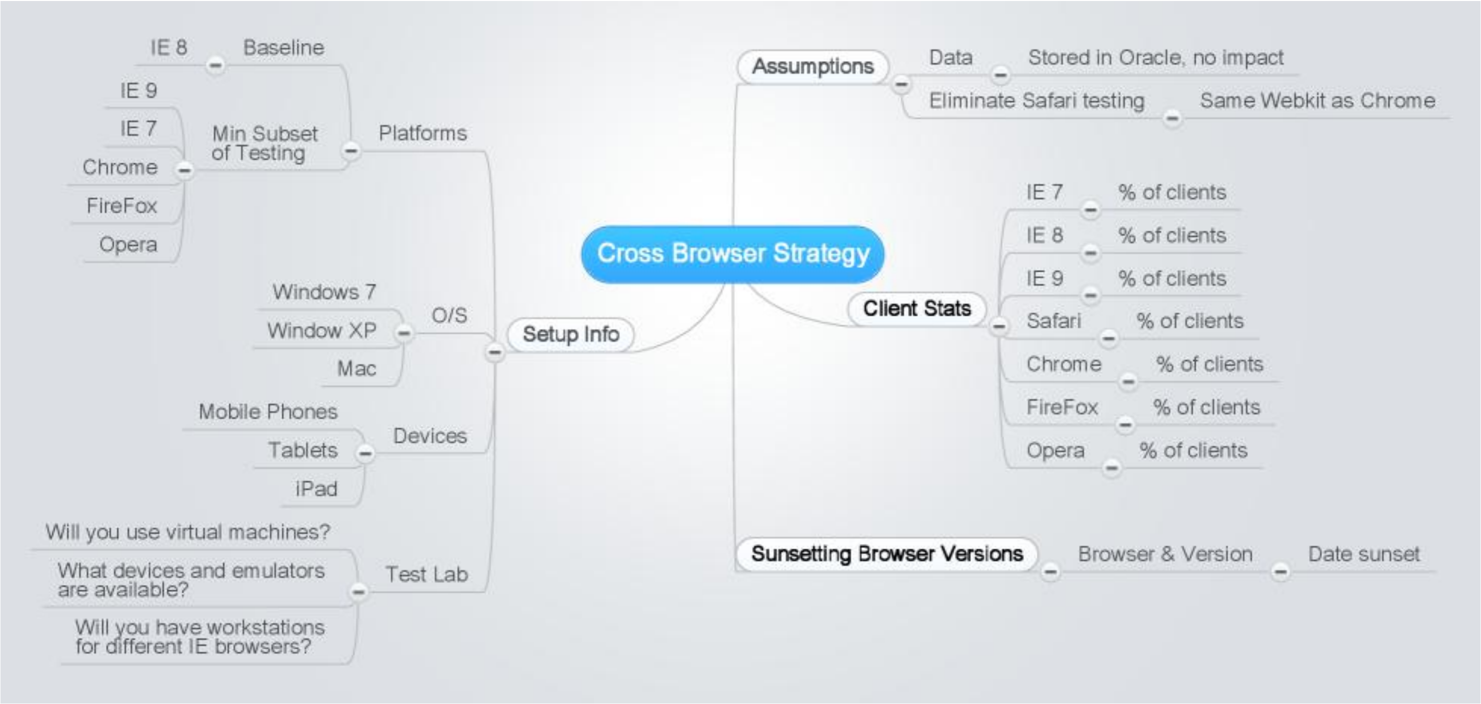
To layout and manage your cross-browser strategy, consider creating a mind map that you can update and modify throughout testing. Plus you can add notes to each node to document additional information. Below are sample mind maps that I created using Mind Meister with a few suggestions to layout your cross-browser testing.

*Cross-Browser Strategy Map*

I like to use a strategy map to capture high-level information that feeds into my testing approaches.

### Setup Info:

The setup information defines the browsers, operating systems, devices, and your test lab. You may want to list your browsers in order of importance for testing. The operating systems and devices can have additional nodes to identify which browsers are tested.



### Assumptions:

What assumptions are driving some of your decisions? Does the intelligence you gathered support your assumptions or help you gain a better understanding? I like to document my assumptions so they can be reviewed with stakeholders and tested when appropriate. It is also good information for future reference when you are asked questions on your decision-making process.

### Client Stats:

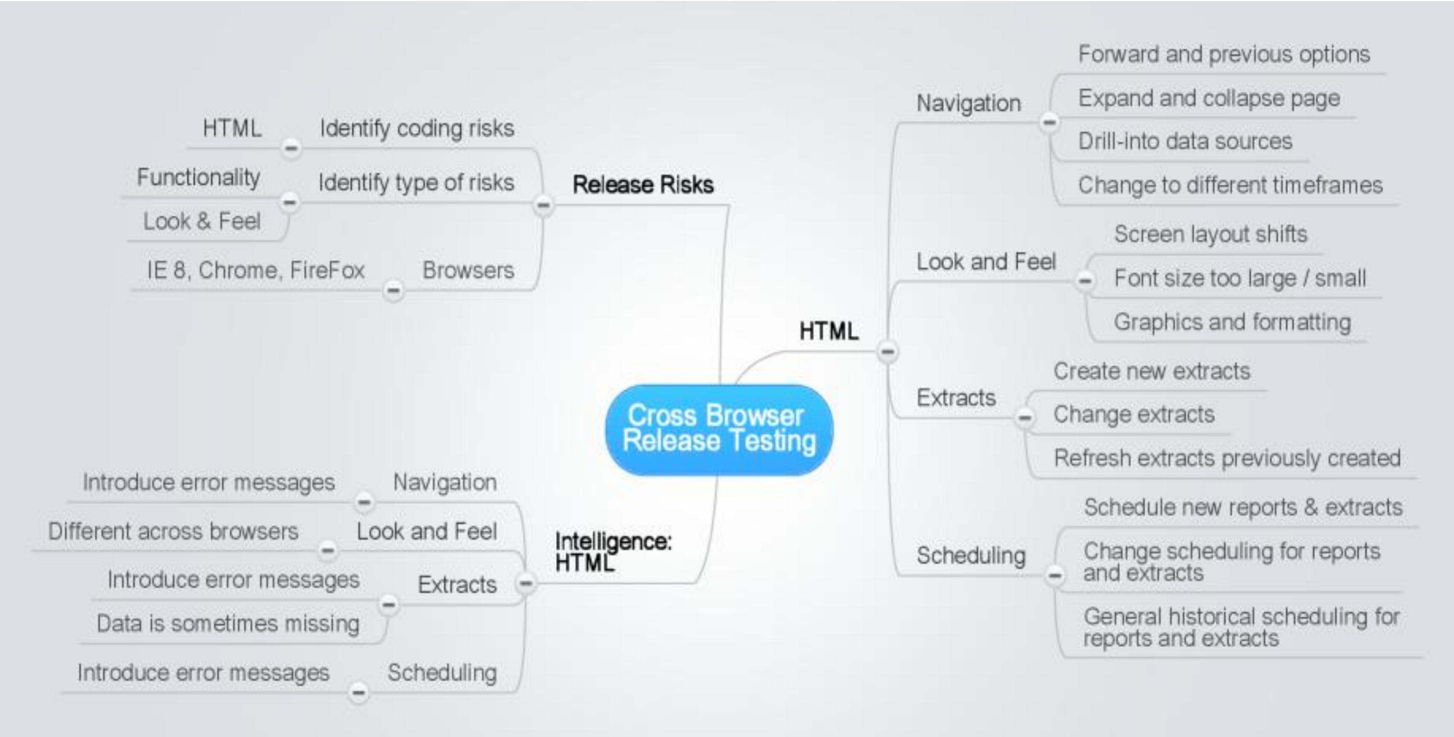
Document the browsers used by your clients and the percentage of usage. Tracking this information can help you make better decisions on how much time to spend testing a browser. Typically a department such as Product Management will determine which browsers and version are supported. However, as a tester you may need to make decisions on how much time to allocate testing different browsers based upon available time.

### Sunsetting Browsers

From a historical perspective, it can be helpful to track information such as when you sunset specific browser versions.

### *Cross-Browser Release Testing*

I like to create a second map that is specific to a release providing general testing guidelines. For a specific release I may create a map with the features requiring cross-browser testing and which tests to perform. I do not provide "how" to perform the test but instead specific areas such as look and feel. The below example provides the type of tests that might be performed for new features using HTML code.



### Release Risks:

Based upon the release you are testing, what coding risks (i.e., html, JavaScript, query, plugins) and what are the areas at risk? For example, you may be concern if specific functionality works across browsers or your concern may be the look and feel. These testing concerns can be different based upon the coding and browsers. I have witness tests that failed only in certain browsers.

### Intelligence:

During your initial cross-browser testing it is important to gather your intelligence on what fails or works across browsers. For example, you may perform testing across different versions of IE to determine if there are any differences. The results of this testing would be documented concisely in this section. You can document additional details in a note field associated with the node. I use the intelligence section for high-level information whereas the compatibility section defines categories of tests to perform. For example, my intelligence indicate there can be a problem with extracts whereas my compatibility testing may indicate to create new extracts, change extracts, and refresh previously created extracts.

### HTML Compatibility:

Defining the minimum subset of testing based upon coding risk is helpful. Listing the tests in order of importance can help the testers if they run low on time and need to scale back testing.

I develop this list as a guideline but I do not define how to perform the tests. This allows them creativity in how they perform their tests and ability to change direction based upon what they witness in testing. Remember to periodically review your minimum subset of testing to determine if changes are warranted.

Back To Index

## In conclusion, if it makes sense, test it

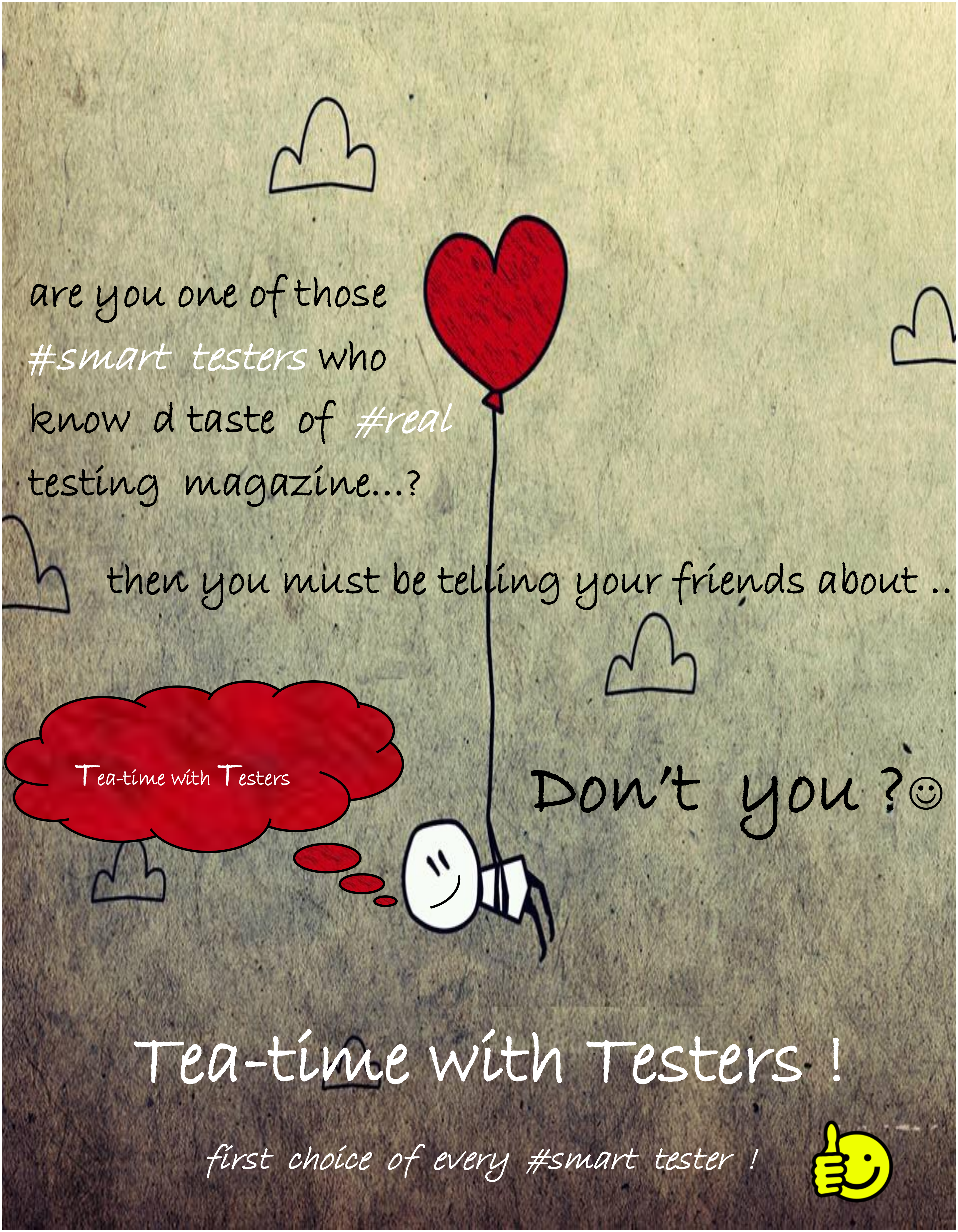
Throughout this series, I have presented a lot of information provided by various testers from strategies to heuristics. I hope this information provides you with some guidelines to get started or evolve your cross-browser testing. However, balance this information with your own knowledge of the product under test, known risks, and do what makes sense. Do not eliminate a test solely because a heuristic (DRY) tells you not to repeat a test if you have a business need to run the test. I use heuristics and other guidelines as a way to challenge my testing ideas but not to dictate how I test. Remember you know your product and risks. Gather information from the Testing Community and determine how or if it applies to your testing. Best wishes with your cross-browser testing and feel free to contact me if I can be of assistance.

**Bernice Niel Ruhland** is a Software Testing Manager for a software development company with more than 20-years experience in testing strategies and execution; developing testing frameworks; performing data validation; and financial programming. To complete her Masters in Strategic Leadership, she conducted a research project on career development and onboarding strategies. She uses social media to connect with other testers to understand the testing approaches adopted by them to challenge her own testing skills and approaches.

The opinions of this article are her own and not reflective of the company she is employed with.

Bernice can be reached at:

LinkedIn:
http://www.linkedin.com/in/bernicenielruhland
Twitter: bruhland2000
G+ and Facebook: Bernice Niel Ruhland


are you one of those #smart testers who know d taste of #real testing magazine...?
then you must be telling your friends about ..
Tea-time with Testers
Don't you ?☺
Tea-time with Testers !
first choice of every #smart tester !

# testing intelligence

*- its all about becoming an intelligent tester*

an exclusive series by **Joel Montvelisky**

## Spring cleaning your testware

In my family we have a tradition, every Spring we systematically go over all the house cleaning, organizing and mostly getting rid of all the things that we don't need anymore.

We do this around the Jewish holiday of Passover, that is set to coincide every year with Spring in the Northern Hemisphere.

But from talking to friends who are not Jewish and don't live in Israel I understood this is more than just a religious-driven tradition. I even found an entry on Wikipedia about it under Spring Cleaning.

## We are better at getting things than getting rid of them

A Universal truth about humans is that we are good at collecting stuff.

We have no problem buying a new t-shirt even if we already have 35 in our closet. Or getting a stack of 50 blank DVDs to burn, even if we have not yet finished the stack of 15 DVDs we bought a year ago, just because there was a sale in our computer hardware store.

We like getting stuff, it's in our nature…

On the other hand, we are really bad at getting rid of it.

When we realize the pants we took out of the closet are not our size anymore 😀 , or that the t-shirt you wanted to wear has been washed so many times it became see-through, we don't immediately throw them away or given them out to charity.

We place them in our closet in the area of "things that I don't think I'm gonna wear soon" and forget about them. Until one day this area of things you are not going to wear takes up half your shelves!

But don't worry, you are not alone! Most of us are just like you.

Nor does this only happen with our clothes/hardware/kitchenware/kids-toys/etc. This happens with everything, including with our testware.

## The art of collecting useless testware

The same thing I just described about t-shirt, DVDs and pants happens also in testware.

When you are looking for a test case to check a feature being updated in your application, you will look for a similar tests for approximately 45 seconds before deciding that you "might as well just write a new test for it" (without realizing there are 2 tests that could have been slightly modified to cover this change).

Or, when part of your application is completely re-written and many of your tests become useless, you don't delete them right away because you may "need to release a patch further along the road". But 2 years later, when there are no patches in sight and none will be created, you still keep these tests because "they are not harming anyone there", and plainly you don't have the time to review them and delete them right now…

If you look into your testware right now, and if you have not done any cleaning lately, it is reasonable to assume that between 10% to 15% of your tests are not relevant anymore, and another 15% to 20% of them need to be updated to match the current state and functionality of your AUT.

## But this is not hurting anyone, right?

## WRONG!

So you have a lot of useless tests, so what?!

It is not like you are paying more to electric company for them! You are also not wasting any extra computers to store them! And as they said countless times lately, storage space is so cheap nowadays that you should not even think about it anymore.

If so, what's the big deal with having useless tests in my testing repository???

The big deal is that you are not looking at the problem from the correct angle. The waste is not in computers, or electricity or storage… these were the expensive resources 10 or 15 years ago. Today the most expensive resources are the human resources – your team – and by having many useless tests in your repository you are wasting their time!

You are wasting the time it takes them to find the tests they need to run. You are making it harder for them to understand if they need to create new tests or if they can re-use some the tests that are already in your Test Management System.

And maybe most importantly, by having a mix of good tests and bad tests in your repository you are increasing the chances of a tester that will run a wrong test and miss part of the issues that a good test would have detected quickly.

## So what can you do about it?

## 3 steps to help you clean your testing repository quickly.

OK, so let's assume you cannot drop what you are doing right now and take 3 days of all your team to go over your testing repository and clean it up. There are still simple things you can do to start improving your process and make it easier for your team to clean up your tests in the future.

### Step 1 – Organize your testing repository

The first step is the one that may provide the biggest immediate impact.

Most times the main problem with your tests is that your testers are not able to find them.

Once this starts happening then they either write duplicate tests (creating more useless stuff in your repository) or choose to run tests informally from their heads (maybe missing part of the things they should be checking).

To stop this from happening you need to organize your tests in a way that will help your testers find them easier and faster.

At the risk of being a little pretentious or immodest, we provide a pretty nice solution for this in PractiTest in the form of our Hierarchical Filter Trees, that let you organize and catalogue your tests in multiple ways at the same time (automatically organizing everything based on your tests' fields).

One of the coolest things of this feature is that it allows you to "place" tests that check more than one area of your product under each of these areas (e.g. a test that checks your Website and your Reports Console and your User Management Module at the same time) instead of having to choose only one area (or folder) for each of your tests. But enough of PractiTest for now…

If you are using any other test tool or even Excel to manage your testing you should try to create the most intuitive and simple classification in your test tree, use a convention that will let you manage test cases and the rest of your test ware.

When choosing a convention, the most intuitive one is usually based on your application's structure. Start from the products or components, and then go down 2 or 3 levels to Sub-Components, Features and even Screens.

Your convention also needs to take into account cases where one of your tests covers more than one product or components (like the case of end-to-end testing scenario we talked about previously). You may choose to look for the "main component" for each test and set it under this categorization, and if this is not clear enough to have a number of "central components" where all the integration tests should reside in.

Conventions will never be perfect, and there will always be cases that fall out of it. But it is better than every tester placing tests where he thinks they should be…

### Step 2 – Mark your tests while you are running them

The second step aims at making it easier for you to do your Spring Cleaning once you have the time for it.

Add a new field or parameter to your tests that each of your testers can set when they are either running or reviewing their existing tests as part of their work, and ask them to them to set in this field one of the following 4 values:

**Ready** – if the test is OK and it can be run as is.

**To Review** – if you think something can be update in the test to make it better. In this case you may also want to add a comment with why you set it to it.

**To Repaire –** when there is something that needs to be changed. In this case too, write down what you think should be fixed.

**Obsolete –** when the test should simply be discarded.

By having these fields you will be able to review them a lot easier. You will also be able to see what tests are "abandoned", by looking for tests that after a couple of cycles still have no value filled on this field. Abandoned tests are usually useless and should also be discarded.

### Step 3 – Set time aside in your schedule before your next "big project"

The last step is the trivial one. You need to set time aside to clean your tests.

The best time (and often enough the only time) you will have is in the short span between projects.

If you plan this ahead and make sure that no one else has already assigned a task for you and your team, you may be able to review all your tests. In case you are not able to go over all of them at least start from the most critical areas of your product.

### Make it a habit for your testers to work correctly

Once you have a clean repository it will be easier for you as test lead or manager to ask your tester to work in a cleaner and more organized way.

Ask your testers not to write unneeded tests, and to always place tests in their correct are within the test tree.

No team will be perfect and with time your tests will again run into a small or larger level of chaos, but at least when you start from a clean page it takes it a little longer to reach this state.

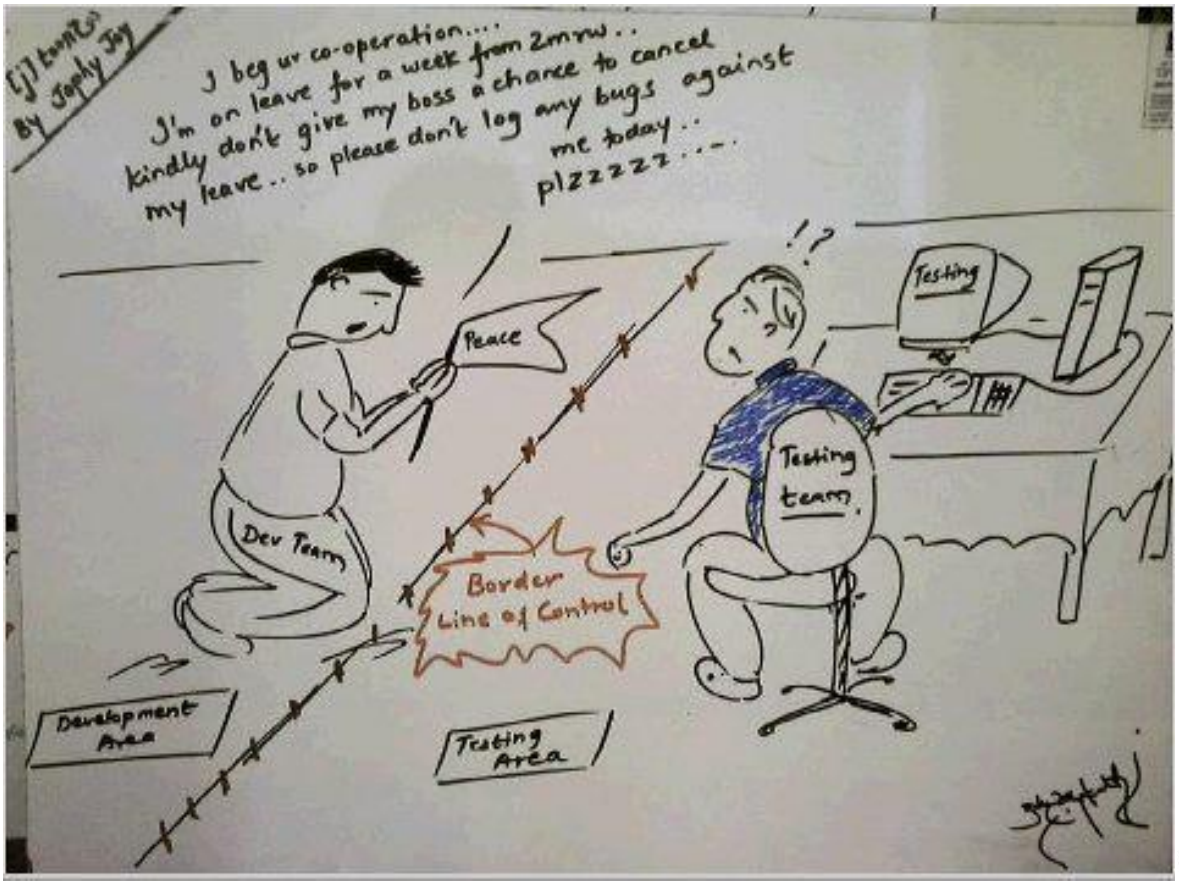
**Joel Montvelisky** is a tester and test manager with over 14 years of experience in the field.

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 joelmonte


[J] toons
By Jophy Joy
I beg ur co-operation....
I'm on leave for a week from 2mrw..
kindly don't give my boss a chance to cancel my leave.. so please don't log any bugs against me today..
plzzzzz...-.
!?
Peace
Testing
Testing team
Dev Team
Border Line of Control
Development Area
Testing Area

Back To Index

# 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...!

**Click HERE to read our Article Submission FAQs !**

sciencephotogallery

## Reflect and change

In the article “Mirror, mirror on the wall, who is the fairest of all?” published in July 2012, the focus was on metrics and how you can use them build a great product. This article continues in a similar line, with emphasis of how these can help you drive change in your behaviour. Change behaviour to do better. The prior article was on how reflections assess the product quality, while this is changing yourself.

I used the wonderful story of Snow White and the Seven Dwarfs in the previous article and am motivated to use it again. In this story, the queen asks the above question everyday to the mirror and feels very happy when the mirror tells her that she is the most beautiful one. And the real story commences when the mirror replies "Snow White is the most beautiful".

A mirror is a wonderful device; it reflects you, it tells you instantly how you look - complexion, girth, emotion, goodness, aberration etc. You can look at the mirror and know what is happening and what action you should take. If for example, your skin looks parched, dry and lifeless, you know that you have hydrate and nourish it with oil. And that is exactly what metrics should do; to reflect the situation clearly so that we can do something positively. And a talking mirror - Wow!

Measurements that we perform and the metrics we collect should be purposeful, to reflect the status - of progress (of testing), quality (of system) adequacy (of tests) and process (aspects like efficiency, productivity, optimality ...).

Just like how the mirror makes us understand the situation and changes how we respond to it by changing our behaviour, good metrics should ultimately result in changing our behaviour.

So what are the behaviours that we would like to change? Being more effective in uncovering defects, being more efficient in testing, and driving behaviour change, to prevent defects.

Knowing the number of test cases and their distributions by features (entities they test), by test types, by quality levels, by positive: negative distributions, reflects if the test cases can be effective. Targeting features that are newer, riskier, complex would imply that the number of test cases should be high in these cases. Distributions by quality levels and test types are called "test depth" and "test breadth" respectively in HBT (Hypothesis Based Testing), and the product of breadth and depth is an indication of test coverage.

To "mirror" efficiency, the typical metric is the percentage of test cases automated, while the interesting one is "immunity" (in HBT), that measures the consecutive pass rate of a test case to ascertain the parts of the system that have become "immune to defects".

Finally the best behaviour change is when fewer defects are detected with lot more prevented. The metrics that can give us an indication are the number and kind of the questions that we ask and the number of potential defect types (PDT in HBT) that are we are looking for.

Capturing metrics that at best are objective indicators of some facets of the system are not useful as they are mere statements of "what of the system", but do spur us into action. So when we measure a 'Defect severity distribution', what does it mean?

So if you want to "spur into action" and change the behaviour you need the "talking mirror". Take a look at the metrics you collect and ask if they change your behaviour.

"Mirror, mirror on the wall, who is the fairest of all?" And the mirror replies "You!" Cheers. Have a great day!

Back To Index

**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**

He has been traveling the world teaching the craft of software testing to many testers. He has specialized in courses on *Rapid Software Testing* – which he co-authored with James Bach.

He is also a prolific writer, and his publications include hundreds of articles, papers and columns.

We spoke with Michael Bolton this time and got to know many interesting things.

Read in this exclusive interview…

# Over a Cup of Tea with Michael Bolton

## We are curious to know about your journey in software testing. Where and how did you start? Milestones reached that you find worth mentioning?

I started testing, as all people do, pretty much the moment I was born. For several years, I tested happily and learned quickly. Then I started going to school.

In school some teachers taught me that questions had one and only one right answer. The activity they called "testing" in school tended to emphasize that. Testing, in school, is mostly focused on confirming that you've learned to provide the right answer to a specific question. I had a very good memory which sometimes help to disguise the fact that I really hadn't learned the material, but that I could remember right answers. To my dismay, we sometimes do software testing in exactly that way, focusing on correctness. I was lucky to have had some teachers— and my parents— who taught me that the world was more complicated than that. Some questions have ambiguous answers. Some answers have lots of viable alternatives. There's more than one way to do things, and there's more than one way that things can go wrong. In light of those ideas, I think we testers have to focus on exploring, discovery, investigating, learning, discussing, reporting, rather than on single right answers. We need to think about what else might be going on in the system, and about looking for problems.

Although I didn't realize it as I was doing it, I prepared myself for a career in software development by working in the theatre. In theatre, people prepare a model of an alternate reality in a few weeks. Talented and sometimes headstrong people figure out how to work together creatively. That turns out to be very similar software development, which combines technical and engineering work with something that, if we paid attention, would look more like art. Though I trained as an actor, I preferred the technical end of things, and eventually became a stage manager. A lot of the skills that I learned there—in particular, self-management, adapting to very rapid change, and coordinating people on a deadline—carried over into software development.

As with many people in testing, I was a programmer before I became a tester. If you learn to program you'll quickly realize how important it is to test your work. It may surprise you, but I believe that most programmers are terrific testers. If you want evidence, watch how they test someone else's code. Like the rest of us, though, programmers have systematic blind spots that allow them to miss subtle but important problems in their own work from time to time. When I was a programmer, I tended to find out about mistakes sooner or later, and the feedback helped me to become better programmer—and a tester.

After a couple of years doing fairly routine programming, I started working for a company called Quarterdeck. I started as a technical support person, working in the Canadian office for three years. I was given my first job in testing by Skip Lahti. He brought me to California and put me in charge of coordinating the testing work on DESQview/X, which was an implementation of the X Window system for DOS (two technologies you don't hear much about these days). After a while I became a program manager for our flagship products QEMM-386 (a memory manager) and DESQview (a multitasker), which at that time were among the best-selling pieces of software in the world. At Quarterdeck, the program manager coordinated the technical aspects of product development, but that always felt like a testing position to me. I was also program manager for the first couple of versions of CleanSweep, which was also quite remarkably successful in its day.

Towards the end of my tenure at Quarterdeck, I tried to go back to being a programmer for a little while, but the program management work kept pulling me back in. Due to the extreme growth and extreme contraction over a couple of years, I left Quarterdeck in July 1998. I've been an independent consultant mostly specializing in testing, ever since.

## Please tell us about your experience with Rapid Software Testing.

Starting in 2001, James and I had met a few times, and in 2004 he asked me to learn how to teach his class, which I did for the first time Bangalore, India. The class really appealed to me, because it was rooted in the kind of fast, responsive work that I had done in commercial software. Meanwhile, at the time, he felt he needed new exercises. I eventually brought the dice game and the Pattern exercise into the class, and I did a lot of work on developing the version of the Wason Selection Task that we teach. In 2006, James made me a full partner and co-author in development of the class, which was very gratifying. We both teach the class in our different ways, and now we've added a third person—Paul Holland—who himself is teaching the class in his own way, informed by his career as a test manager. It's important, we believe, that each tester owns his or her own approach to testing, so we give each other a lot of freedom and latitude to try things, to learn things, and to develop new exercises.

That's a very creative process, and it's very strongly influenced by the participants in the class, too. We learn a ton from them.

## People who know very little about RST often ask, 'How to follow RST? What different things testers do (as compared to traditional way of testing) when they do it the RST way?' Would you help our readers understand RTS in brief?

Rapid Software Testing is fast, inexpensive, credible, and accountable, intended to fulfill the mission of testing completely. If you want a good description of Rapid Testing in a nutshell, have a look at the map here (http://www.satisfice.com/images/RapidTestingFramework.pdf) and the premises of the class detailed on my blog, starting here .

Rapid Testing is focused on software development as a human activity. It's centered not around documents or processes but on the skill set and mindset of the individual tester. Testing sometimes generates artifacts, but testing is not about the artifacts. It's like music: you can have sheet music, which notates music would be like if it were played, but until it is actually performed no music happens. So Rapid Testing emphasizes the tester performing the testing. If it's important to do other stuff—if it's part of the mission—we do that. If we're asked to produce comprehensive documentation, we do that. If we are asked to test within a particular context, or to do highly formal testing, we do that. But we'll always question the cost and value of what we're doing, because we're always focused on trying to find important problems quickly, to add clarity, to reduce waste, and to speed up the successful completion of the product.

## What is your opinion on Test Metrics and their usefulness? Are there any good or bad metrics? Why do Executives ask for it? And how should testers/test managers tackle such demands?

Anthropologists have told us that when people feel in the presence of forces that they can't control and that they don't understand, they resort to magic and ritual in order to try to get a feeling of control of them.

So when managers and executives don't understand how to make the product better, or how to make the development, better, or how to make the testing better, they ask for numbers, and then hope that they can make the *numbers* look better.

Most managers and most testers haven't studied measurement, and don't pay much attention to the question of whether they're measuring what they think they're measuring. Moreover, they don't recognize the effects that measurement can have on motivation and behavior. To get started on addressing the first issue, read "Software Engineering Metrics: What Do They Measure and How Do We Know" by Cem Kaner and Walter P. Bond (it's easy to find on the Web). In that paper, there's a list of ten questions that you can ask to evaluate the quality of the measurement you're making. Read the material cited in the paper, too. And read *Quality Software Management Volume 2: First Order Measurement* by Jerry Weinberg. For the second issue, have a look at *Measuring and Managing Performance in Organizations*, by Robert Austin.

Quite frankly, I think testers have been very bad at explaining their work and how it's done. That's partly because we're in a fairly new craft. The craft has also been stuck in ideas that were first raised somewhere in the late 1960s and early 1970s. These ideas gave rise to the idea of the testing factory; that testing is an ordinary technical problem; that testing is about verification and validation; that testing is about checking to see that something meets expectations.

I resist that really strongly. Testing is far more than determining whether something meets expectations. Testing means telling a story about products, the kind of story an investigative reporter might tell: learning about products, investigating them, building new questions from that knowledge, and letting the expectations fall where they may. Some think of testing in terms of a game: preparing a set of test cases that the program has to pass, and then keeping score as to how many are passing and how many failing. Testing isn't a game; it's an investigation. Numbers can illustrate the story, sometimes, but the numbers aren't the story.

## How should testers develop their Critical thinking? Any tips?

There are a bunch of different books you can read tools of critical thinking by David Levy or thinking fast and slow by Daniel Kahneman but in order to get really good critical thinking. You have to practice it you have to practice asking yourself what else could this be you have to practice asking yourself how might I be fooled. You have to hang out and invite challenges from other people. Critical thinking is really thinking about thinking without being fooled with the object adopting fooled.

## What are the things about current state of software testing that make you happy and things that worry you?

A few years ago it was difficult to find people who are interested in doing software testing really well and really skillfully. That was true in India and it was true in the rest of the world too. These day things are very exciting. We're discovering that there are lots of people who are interested in sharpening their skills in taking an expansive view of what testing is and what could be. I'm also very excited about the fact that some large institutions are starting to pay attention.

I'm less excited about the fact that there are still people who want to trivialize our craft by selling bogus certifications and holding the threat of unemployment over the unwary. I'm still surprised at the number of people who don't want to study testing, and the things that surround it—how we make mistakes, how we model, how we learn, how we can use technology productively and expertly, how we can think critically. We've got fabulously interesting jobs.

## We know people who want to follow RST and CDT approach in their project works. But they can't since they don't have support from management. Organizations (most) treat experiments and risks equally. What would you advice them?

Rapid software testing is not focused on the organization. It's focused on the mindset and the skill set of the individual tester. You can choose to test rapidly in any context, mostly by exploring, discovering, investigating, learning and reporting about the product. One of the big deals in rapid testing is optimizing your own work. You can do this yourself, in your own little corner of the world, in those little snippets of unsupervised time that you can afford to waste without getting into trouble. We call this "disposable time". If your management is doing things that are wasteful and unhelpful even to them, they're probably not paying enough attention to notice that you doing anything differently anyway. On the other hand, they might notice when you're providing more information and more valuable information them. So try little experiments. Share ideas with other testers in your community or your organization. Keep track of how much time you're spending on excessive scripted work. Learn to use lightweight, flexible tools, and study some basic programming. Write little tools that save you time or tedium. Read. Read anything and connected to software or testing. Develop your skill with Excel, or with databases. Keep track of how much time you're spending on test design and execution, on bug investigation and reporting, on setup, and administrivia work. Make that visible; graph it out.

Organizations that regulate themselves to the degree that they're unwilling to take some little risks or tolerate small experiments will tend blow up or be surpassed by competition. Look at The Black Swan and Anti-Fragile by Nassim Taleb for some terrific ideas along those lines.

## 'Value Add makes client happy' is common belief in testing circle. What #value do you think testers can add to project that will make customer happy and how?

I've never heard the expression, "value add makes client happy", so I don't exactly know what it means. Much of the value that testers add to the project is often indirect and, for some people, hard to observe. Here's what we do add: awareness of what's going on, particularly in the product, but also in the project and in the business.

I sometimes hear people complain that the testing is the bottleneck, saying things like "we can't ship the product until the testing is done." I disagree; they could ship the product whenever they liked. I think what they really mean is that they can't ship the product until they believe the development work is done—and can't decide whether it is done. So it's not that there's a bottleneck; the problem is that no one knows what's really in the bottle. Testing helps to address that problem.

## We watched you acting in ‘The Angel and Devil of Testing’. Where did the idea come from? Would you like to share that experience with our readers?

I don't exactly remember how the angel and devil piece got started. I remember doing a talk called "Two Futures of Software Testing" in which I played the devil side for the first half and something more like an angel for the second half, but I did that on my own. I believe Lee Copeland put Jonathan Kohl together with me to do the two-handed presentation. Since Jonathan's outward appearance is more angelic than mine, I got to play the devil. Jonathan's wife Elizabeth did some terrific graphics for it, too.

## What is your opinion about Tea-time with Testers? Any message for our readers?

I think Tea Time with Testers is a wonderful initiative, and congratulations to you for starting it.

To your readers: Read on, but read critically. And when you notice something worth writing about, write!

# OUR PARTNERS

## Quality Testing

Quality Testing is a leading social network and resource center for Software Testing Community in the world, since April 2008. QT provides a simple web platform which addresses all the necessities of today’s Software Quality beginners, professionals, experts and a diversified portal powered by Forums, Blogs, Groups, Job Search, Videos, Events, News, and Photos.

Quality Testing also provides daily Polls and sample tests for certification exams, to make tester to think, practice and get appropriate aid.

## Mobile QA Zone

Mobile QA Zone is a first professional Network exclusively for Mobile and Tablets apps testing.

Looking at the scope and future of mobile apps, Mobiles, Smartphones and even Tablets , Mobile QA Zone has been emerging as a Next generation software testing community for all QA Professionals. The community focuses on testing of mobile apps on Android, iPhone, RIM (Blackberry), BREW, Symbian and other mobile platforms.

On Mobile QA Zone you can share your knowledge via blog posts, Forums, Groups, Videos, Notes and so on.

by

Quality is delighting customers
QUALITY TESTING

# Testing PUZZLES by Sebi

Claim your Smart Tester of The Month Award. Send us an answer for the Puzzle and Crossword bellow b4 20th April 2013 & grab your Title.

Send -> teatimewithtesters@gmail.com with Subject: Testing Puzzle

## “Find it out”

List the maximum number of ***.swf** files that you can find located on microplace.com domain (any subdomain is valid)"

Back To Index

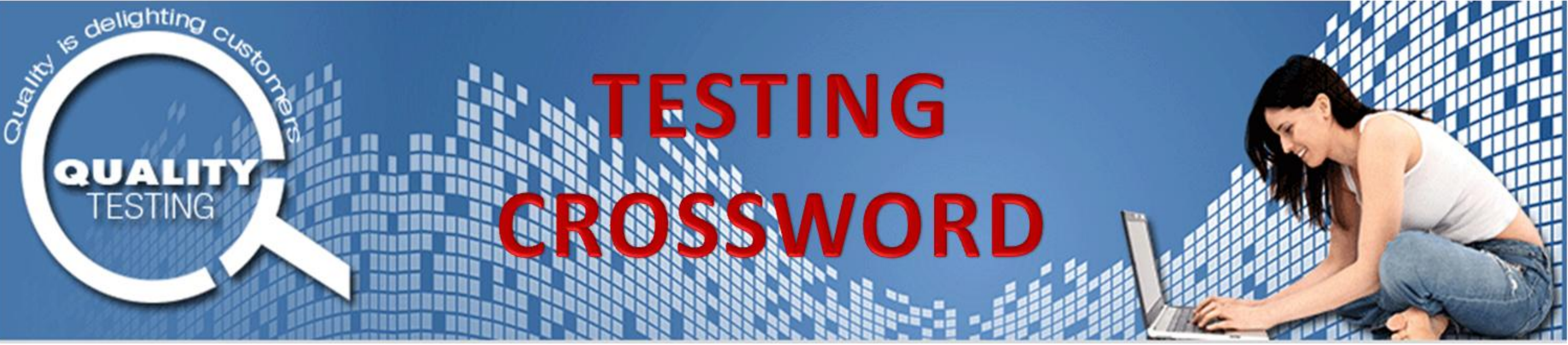
## Biography

**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



# TESTING CROSSWORD

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | | 2 | | | 3 | | 4 |
| | ■ | | ■ | ■ | | ■ | |
| 5 | | | U 6 | | | ■ | |
| | ■ | ■ | | ■ | 7 | | |
| 8 | | 9 | | | 10 | ■ | |
| | ■ | | ■ | ■ | 11 | | |
| | 12 | 13 | | 14 | | ■ | |
| | ■ | | ■ | | ■ | 15 | |

**Horizontal:**

1. It is a web-based test case management tool (8)

5. It is an automation tool to test graphical user interfaces using images (6)

7. It is designed to transport and store data (3)

8. It is an open source Load Testing solution that is free and cross-platform (6)

11. After the designing and coding phase in Software development life cycle, the application comes for testing then at that time the application is started as ______, in short form (3)

12. It is an open source, free to use operating system widely used for computer hardware and software, game development, tablet PCS, mainframes etc. (5)

15. It is the process of examining written code with the purpose of highlighting mistakes in order to learn from them, in short form (2)

**Vertical:**

1. It is a web based test management and execution tool, includes test specification, planning, reporting, requirements, tracking and collaborate with well-known bug trackers (8)

2. A set of activities conducted with the intent of finding errors in software, in short form (2)
3. It is an operating system commonly used in internet servers, workstations and PCs by Solaris, Intel, HP etc. (4)

4. A device, computer program, or system that accepts the same inputs and produces the same outputs as a given system (8)
6. Testing of individual software components, in short form (2)

9. A formalized set of software calls and routines that can be referenced by an application program in order to access supporting system, in short form (3)

10. Assessing the effect of a change to an existing system, usually in maintenance testing, to determine the amount of regression testing to be done, in short form (2)

13. The individual tested units are grouped as one and the interface between them is tested, in short form (2).
14. The specification of tests that are conducted from the end-user perspective, in short form (2)

# Answers for last month’s crossword:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| N | G | R | I | N | D | E | R |
| T | ■ | E | ■ | ■ | R | ■ | T |
| I | N | C | I | D | E | N | T |
| M | ■ | O | ■ | E | ■ | ■ | M |
| E | ■ | V | ■ | P | A | T | H |
| J | T | E | S | T | ■ | D | ■ |
| S | ■ | R | ■ | H | ■ | T | ■ |
| W / S | ■ | Y | S | M | O | K | E |

*We appreciate that you*

*“LIKE” US !*

Join us on Facebook.

You are just a CLICK AWAY

Every Tester
who reads Tea-time with Testers,
Recommends it to friends and colleagues .
What About You ?

Image : vernhart

# Our Testimonials

◀ ▶

Dear Tea-time with Tester,

I wish I could send you guys some flowers ☺. I'm writing you to wish you a very happy second anniversary and convey big thanks on behalf of my entire team.

We are not much active online but my team religiously follows your magazine and the articles that you people have been publishing are really, really helpful.

Congratulations once again and we wish you many more years of success.

- Nigel K.

# in nexxt issue

articles by -

Jerry Weinberg

Johanna Rothman

T Ashok

Joel Montvelisky

Ben Kelly

Leah Stockley

# our family

## Founder & Editor:

Lalitkumar Bhamare (Mumbai, India)

Pratikkumar Patel (Mumbai, India)

Lalitkumar

Pratikkumar

## Contribution and Guidance:

Jerry Weinberg (U.S.A.)

T Ashok (India)

Joel Montvelisky (Israel)

Jerry

T Ashok

Joel

## Editorial | Magazine Design | Logo Design | Web Design:

Lalitkumar Bhamare

Image Credits- Olive Cotton

## Core Team:

Anurag Khode (Nagpur, India)

Dr.Meeta Prakash (Bangalore, India)

Anurag

Dr. Meeta Prakash

## Testing Puzzle & Online Collaboration:

Eusebiu Blindu (Brno , Czech Republic)

Shweta Daiv (Mumbai, India)

Eusebiu

Shweta

## Tech -Team:

Chris Philip (Mumbai, India)

Romil Gupta (Pune, India)

Kiran kumar (Mumbai, India)

Kiran Kumar

Chris

Romil

*|| 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

Give Feedback