THE INTERNATIONAL MONTHLY FOR NEXT GENERATION TESTERS

Tea-time with Testers

December 2012 | YEAR 2 ISSUE XI

James Bach

Cover Story

The Rise of The Intellectual Indian Tester

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) 9960556841

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

Editorial

Eyes wide OPEN

On the evening of 31st December, I was waiting for a friend in my apartment's lobby. Tired of waiting for him I took out my phone and started playing *Angry Birds*. My society's watchman was just passing by and sound coming out of my phone made him curious.

With that curiosity the guy came to me, observed me playing for a while and asked if I could give him one chance to play. Without hesitation I gave him my phone and also helped him with the controls etc. I didn't fail to observe the excitement on his face while he was playing the game. He was a fast learner and quickly got command over the game.

In the end Sudhir (his name) asked me with same curiosity, "Sir, how much is the cost of this phone?" I told him the price. His face which looked bright earlier suddenly turned pale and he said that he needed to go. When I asked he managed to smile and said, "That's my six months' salary, sir!"

I had no answer and I let him go. Sudhir was gone but he unknowingly made me think about the incident.

That day I learned that richness is a *relative* thing. Most of us want to become rich, richer or may be richest and there is nothing wrong about it. The thing is, we only think of getting rich but most of us never think *'as compared to whom?'* we want to.

And same applies for becoming great at things too. One can be a good tester than his colleague or many other testers at his work place. But should that be enough? How about getting better than those testers who have earned name and reputation in global testing community?

There was a time when I used to say, "I want to be a better tester." But *that* better had no meaning because I had no clue *as compared to whom* I wanted to be better.

Call it my luck or reward of my work but I'm glad to tell that I got to spend a whole week with James Bach this December. Yes, The James Bach. Apart from learning lessons from his Rapid Software Testing class I learned a lot many things from him.

And most importantly, *now* I know *as compared to whom* I *wish* to be a good tester.

In this 2013 and for many years ahead, I will work hard to reach there. As *compared to whom* are you planning to be better this year, by the way?

Wish you a great 2013 with lots of success.

Yours Sincerely,

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

We love it when you write to us.

To send your letters, write to us at

editor@teatimewithtesters.com

November 2012

Dear Lalit,

I love what you've done with IF. It's an excellent guide for testers. I grew up with a copy of Kipling's poem on my bedroom wall. It's particularly good that you took out the references to "men." That was fine in the 19th Century, but the testing profession would be in awful shape if it was limited to men.

I'm particularly pleased that "intelligence" has turned out to be a theme of all the articles in the issue. I know that all the graduates of our Problem Solving Leadership workshop will really appreciate that theme. Next to the attributes of character as captured in your version of "IF," intelligence is the next essential attribute of the best testers.

Ben Kelly makes it clear that Zombies possess no intelligence, or if they do, they don't use it. And having intelligence that you don't use is, I think, even more stupid than just being stupid in the first place. Thanks, Ben, for point out what we often overlook, as when we believe that a tester does enough just to be smart yet not use those smarts.

Intelligence is not something simple, nor is it something possessed only by individuals. Jansson and Nolmark provide something of a menu for an intelligent team. I hope all your readers read their menu and ask themselves how they and their team measure up on this recipe for intelligence. I hope future issues have more examples of intelligent teams–and how they got that way.

Joel M. is right on target, as usual, when he relates "QA intelligence" to "military intelligence." He promises to tell us more in future issues. I'm looking forward to his keeping his promise.

And then, T Ashok winds up this remarkable issue by proving an extremely specific example of one approach to intelligent problem solving. One of many, I think, and I hope we'll see more approaches in the coming year's issues.

- **Gerald M. Weinberg**

QuickLook

Testing Puzzles

by Sebi

Highest profile software failures of 2012

By net-security.org

SQS compiled a list of the worst software failures over the past 12 months. This year's annual survey is based on major software failures throughout 2012 and highlights the continuing problems faced by the financial and banking sector, which have dominated the software glitch top ten lists over the past three years.

In the 2012 survey, financial services software glitches represent five of the top ten. Legacy systems in banks and trading firms are not being updated or replaced due to financial constraints and this is one major cause of failure.

1. Software glitch costs trading firm $440million in 45 minutes

A trading firm's newly-installed software resulted in a $440 million loss after it rapidly bought and sold large volumes of over a hundred different stocks in 45 minutes using a flawed software algorithm that bought the shares at market price then sold at the bid price - instantly losing a few cents on each trade. The rapid trades pushed the price of the stocks up, resulting in spectacular losses for the trading firm when it had to sell the overvalued stocks back into the market at a lower price.

2. Leading securities markets' operator

A stock trading business launching its initial public offering on its own trading system was forced to withdraw its IPO after an embarrassing computer glitch caused a serious technical failure on its own exchange. A system problem occurred as soon as the exchange tried to open the ticker symbol of the stock, failing to roll into a continuous trading pattern as it was supposed to, halting the trading on the stock before it had even started trading.

3. Stock Exchange IPO trading of social media giant falls flat

Technology problems affected trading in millions of shares of a popular social media website, after software glitches caused a malfunction in the trading system's design for processing orders and cancellations, meaning orders were processed incorrectly, if at all. Trades in as many as 30 million shares were affected by the glitch.

4. US elections' vote glitch sees nomination problems

Computer problems drew complaints across the US during the 2012 elections, with numerous problems with voting machine glitches reported by voters. An example was touch-screen errors automatically changing the vote from one candidate to another and not allowing voters to reselect or correct the error.

5. Airline's software glitch strands travelers for the third time

For the third time in 2012, a computer glitch wreaked havoc on thousands of travelers with a US airline, delaying flights for hours. A glitch in the dispatch system software resulted in hundreds of delayed flights across the US and internationally. The two hour outage held up 636 of the 5,679 scheduled flights and resulted in 10 flights being cancelled altogether.

6. Security staff shortage at international sports event

An internal computer systems problem resulted in miscalculation of the number of security staff required to support an international sports event this summer. This internal staff rostering glitch resulted in members of the armed forces being drafted in to act as security staff.

7. Teething problems for new revenue service software system

After upgrading its software and revenue service system, at an estimated cost of $1.3 billion through 2024, to promote e-filing of tax returns, the US revenue service saw delays in handling electronic tax returns, with 85 per cent of refunds delayed by 23 days+.

8. Gambler loses winnings to computer virus

A gambler, who was under the impression he'd won just over $1 million, was told by a High Court that, despite his anticipated windfall showing in the online game he had played, he was not a millionaire after all. A software error mistakenly reported his winnings as much higher than they actually were and, due to this contingency being covered in the game's terms and conditions, he could not legally claim his anticipated prize.

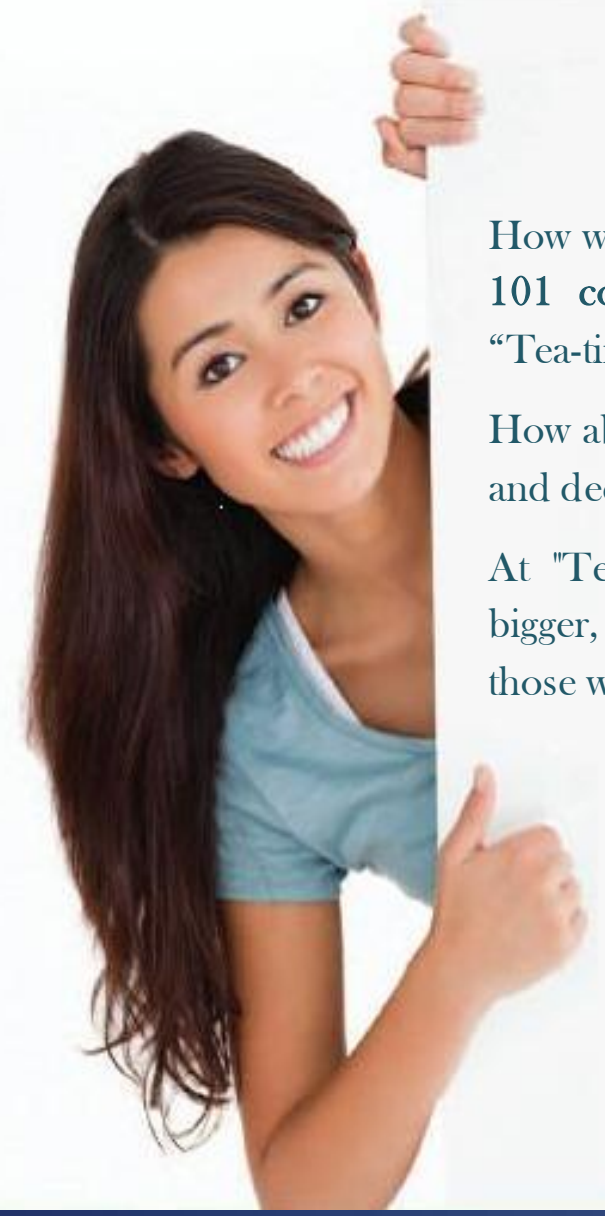
9. Utility customers in the dark over late notice and incorrect payment charges

An Australian energy company sent thousands of customers late payment charges for bills they didn't receive due to a computer glitch, while a Germany utility company overcharged 94,000 of its customers due to a computer glitch that incorrectly charged exit fees, costing the energy supplier $2.24 million in settlement payouts.

10. Leap year bugs disrupt banking and healthcare payment systems

A leading multinational corporation's cloud computing service outage, which affected Governments and consumers, was caused by the additional day in February this year. The same leap year date bug also affected an Australian payment system used by the health industry, resulting in 150,000 customers being prevented from using private health care cards for medical transactions for two days.

Want to connect with right audience?



How would you like to reach over **19,000** test professionals across **101 countries** in the world that read and religiously follow “Tea-time with Testers"?

How about reaching industry thought leaders, intelligent managers and decision makers of organizations?

At "Tea-time with Testers", we're all about making the circle bigger, so get in touch with us to see how you can get in touch with those who matter to you!

ADVERTISE WITH US

To know about our unique offerings and detailed media kit

write to us at **sales@teatimewithtesters.com**

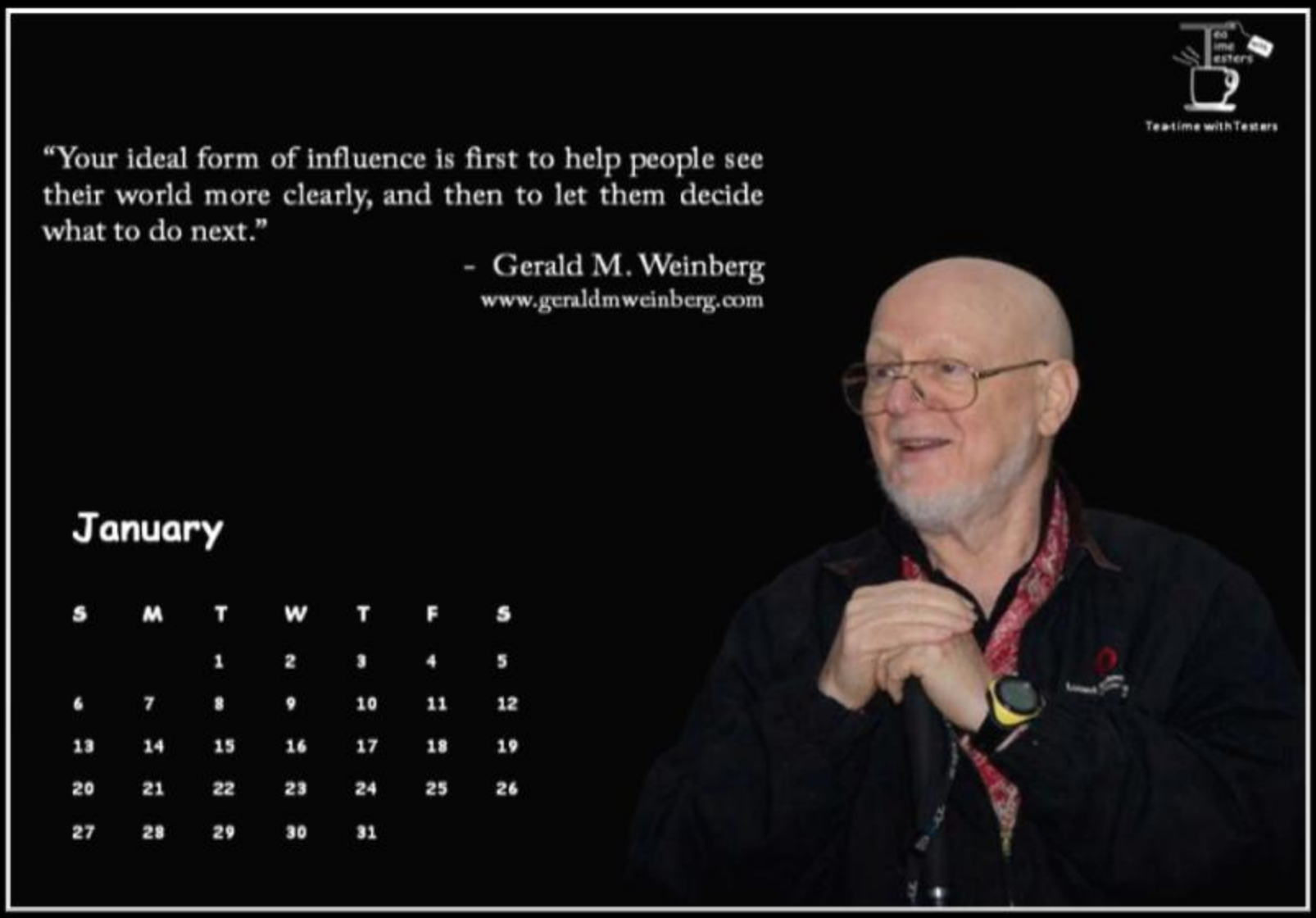
Announcing...

Smart Tester of the Month Award !!!

- Saritha Prakash
- Sreenath Suman Kumar

Congratulations !

Announcing
TTwT 2013 Free Gift Calendar



Click HERE to download

Intelligence, or Problem-Solving Ability (Part 2)

FACETS OF PROGRAMMING INTELLIGENCE

Adaptability, then, is required for all sorts of intelligent behavior. Behavior which, though mental, only requires carrying out a set of fixed rules is not properly considered intelligent. It might better be carried out by machines than by people. Not that carrying out a set of fixed rules cannot be an important part of intelligent behavior. On the contrary, a programmer who cannot add two numbers together without extraordinary difficulty is heavily handicapped in the race for better problem solving, unless, of course, he turns his handicap into an asset by developing shortcuts that bypass the arithmetic he cannot do.

To a great extent, problem-solving technique is idiosyncratic, if only because certain people can do certain things better than others. Each person, if he is intelligent, tends to look for methods of solution that depend on his best qualities and avoid his weakest. As a specific example of such a quality, consider the facet of memory.

There is no doubt that memory is one of the most important aspects of intelligence for a programmer—if he can but harness it. Memory helps a programmer in many ways, not the least of which is by enabling him to "work" on problems when he does not have all his papers in front of him. Consider this anecdote related by a programmer about how he solved a problem while lying in bed:

The problem was given to me by a programmer I encountered yesterday morning at the computing center. He said he had the problem since last April (it is now January). The problem was not very serious, but it had puzzled him on and off since then, and everything he tried failed to work. It was a PL/I program, and the trouble was in the format of the output. He had established an ON ENDPAGE unit, but it only worked at the end of the first page and when it was raised by SIGNAL. After the first page, the listing just went on from page to page without producing the headings he wanted.

I checked the PAGESIZE he was using, and his job control cards—to see if he had some strange carriage control situation. I checked the position of the ON-unit, but obviously it was executing once. Were there any switches in it? Nothing. The only unusual thing I found was that he had used a PUT SKIP to print the heading, not a PUT PAGE as I ordinarily do. I pointed out to him that this explained why the heading he had printed failed to go on the top of the page. But, after testing a number of other hypotheses, I knew I wasn't going to find the major trouble. I showed him how he could get the right output, using a test of LINENO to raise ENDPAGE.

That satisfied him, but l knew l couldn't put my mind at rest until I understood what was happening.

I put the problem out of my mind, but when I went to bed last night, my mind seemed clear, so I decided to work on it as I lay there. I reconstructed the whole situation mentally—the coding and the output. When my eyes are closed and I am in a quiet place, I can call up the picture of any program I am currently working on— even one like this, which I had seen only once. I scanned the output in my mind and tried to imagine what kind of program would produce this output. After reviewing my hypotheses from the morning and rejecting each on carefully considered grounds, I decided to look for something new.

The strange element, I felt in scanning over the program, was the PUT SKIP in the ON-unit. This stood out in my image the more frequently I scanned it, for I never did that. Never? In that case, perhaps it was causing the difference. But why would starting a page in a different way cause the end of the page to be missed? Well, how is the end of a page detected? By being so many lines from the top. But how is the top determined? I realized I didn't know that answer precisely, so I speculated on possible alternatives. By this time, I knew I was on the right path, and I simply considered each alternative in turn, imagining the action each would produce in this program.

Finally, when I worked through the action under the hypothesis that only PAGE (or possibly LINE, which did not apply in this case) could start a new page, I realized the problem. PUT SKIP in the ON-unit did not start a new page—the line number simply kept increasing and no new end of page was ever reached because no new page had ever been started. Satisfied that I had solved the problem, I went immediately to sleep. This morning, I made this test case to demonstrate my conjecture, and you can see that it is precisely as I say.

Without the aid of a fine memory, this programmer might never have solved this problem—and learned something new— because he might never have seen this program again. On the other hand, if he did not have this kind of memory, he probably never would have attempted this approach to the problem. He might, instead, utilize his cleverness at creating critical test cases to solve the problem using the computer. For his inadequate memory, he would substitute an actual copy of the problem program—which is nothing to be ashamed of.

Indeed, by attacking the problem on the machine, this poor-memory programmer might have the solution to the problem before going home, leaving his sleep untroubled by bugs. Which method is superior? We really cannot give an answer in isolation. If, for example, machine access is poor, the second programmer will be at a disadvantage. If, on the other hand, everyone is working overtime and

barely has time for sleep, let alone quiet reflection, the first programmer will never get to show his brilliance. Naturally, it would be best to have both abilities to an equal extent and to apply each as is appropriate to the problem and the overall situation. Short of that, we must make the best of what we have.

Just as different working conditions favor the application of different forms of intelligent behavior, so do different programming phases give different programmers a chance to shine. For example, when we are making the overall design of a program, what we most need is the ability to create new programming ideas and to screen them on the basis of broad principles. Examples of such screening ideas are symmetry of structure and generality of function—one leads to simple coding of difficult problems and the other leads to the solution of difficult problems with simple coding. Still, these critical abilities are useless if there is a paucity of ideas to which to apply them. "Nothing" cannot be criticized. Thus, a programmer lacking in either ability—creativity or selectivity—will be handicapped in attempting to design programs.

When coding, however, different abilities come to the fore. Instead of the broad, sweeping mind, the mind which is clever at small things now excels. Then, when testing, the programmer must switch to yet another group of gifts—particularly the eye for wholeness, or gestalt.

Consider the following tale:

I was eating breakfast and reading an article by Stephen Spender called "The Making of a Poem." On page 120,I reached the end of one section and set the book down to put some more sugar on my cereal. When I picked the book up again to start reading a section called "Memory," I immediately had a feeling that there was something wrong in the first sentence which started:

"If the art of concentrating in a particular way ..."

I felt, more or less simultaneously, that the trouble was in the word "particular" and that it involved a misprint.

The misprint, however, was rather confusing for I sensed that it was a letter inversion but I also sensed—a little bit more weakly, though, I have a definite impression of that—that there was a letter missing. I examined the word "particular"—which, by the way, I often mistype as "particluar"—first for the inversion and, failing to find that, for the omitted letter. I spent a rather long time looking for the error—I could measure that because I ate five or six spoonfuls of cereal in the process, the amount I ordinarily eat between sips of water. But I could not find anything wrong, and when l reached for the water glass, I was rather confused.

The water glass was empty, so I set down the book and went to the sink to fill it. Upon returning, I drank some water, picked up the book, and started to read. I finished the paragraph without further difficulty, but when I commenced reading the next, I immediately saw that the line:

"All poets have this highly devolped sensitive apparatus ..." contained the misprint of the word "devolped," which stood in the same position in that sentence as "particular" had stood in the first sentence of the preceding paragraph. I had the right impression, but I had "focused" wrongly.

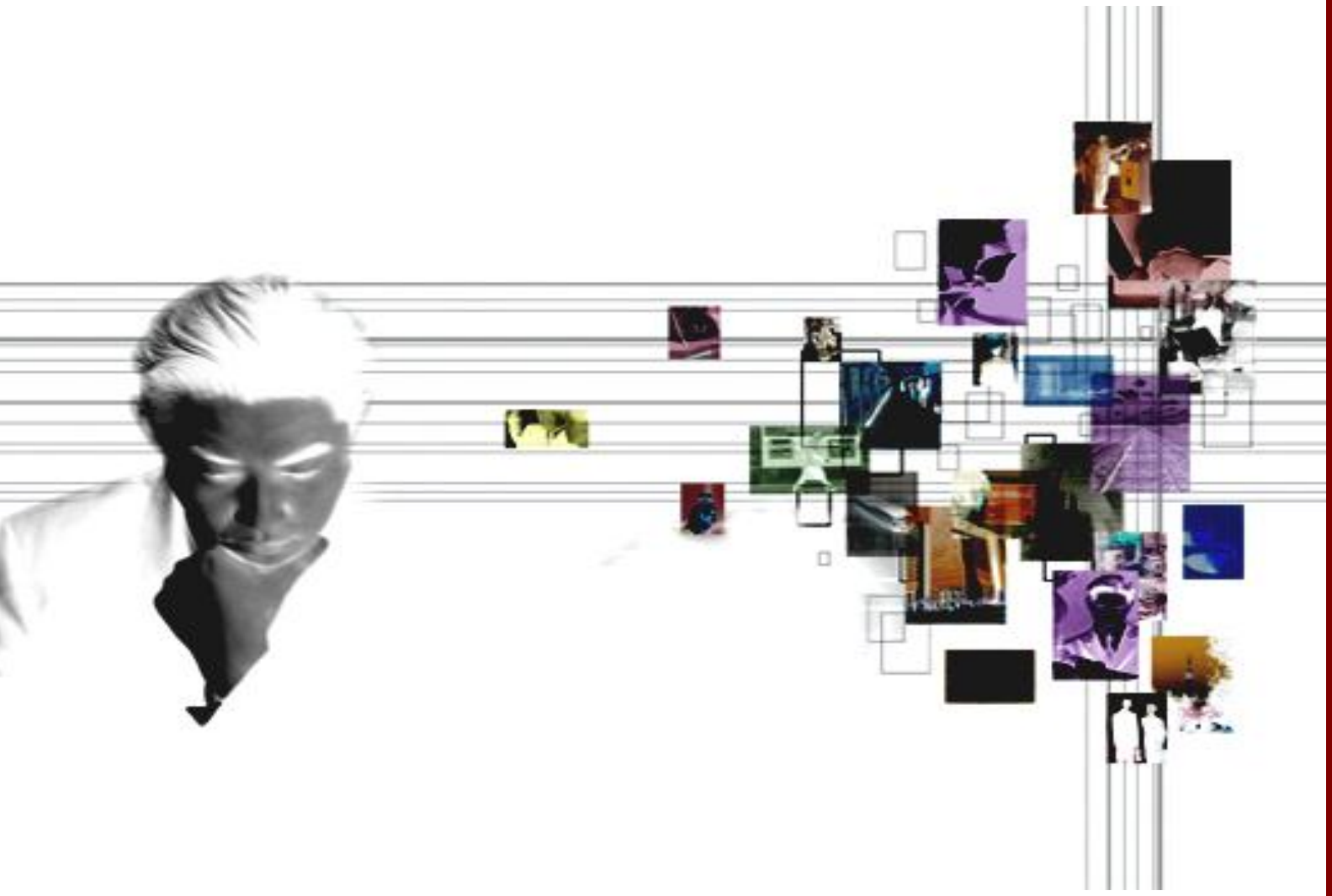
Although this error was in printing, its discovery and location followed very closely the process by which many programming errors are found. First, there is only the gestalt, a general feeling that something is out of place without any particular localization. Then follows the ability to shake loose from an unyielding situation—the ability to change one's point of view, even by employing external devices such as going for a glass of water. Then, however, one must go from the general to the particular—

"focusing," as it was called here. Although one does not find errors efficiently by a detailed search of each line, or word, or character, the ability to get down to details is essential in the end.
Thus, for debugging, an almost complementary set of mental powers is needed. No wonder good debuggers are so rare!

Even more of a rarity is the good documented Documentation difficulties come from many sources. In the first place, if the program is not well written, there is not much that documentation can do to resuscitate it. Since programmers usually document their own productions, the good documenter has to be a good programmer to begin with—and then he must add the capacity to express himself verbally and graphically. Finally, he must have the patience to work out those last few ambiguities in his documentation—for the last 5 percent of the work makes the document 100 percent better.

to be continued in next issue...

Back To Index



Biography

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

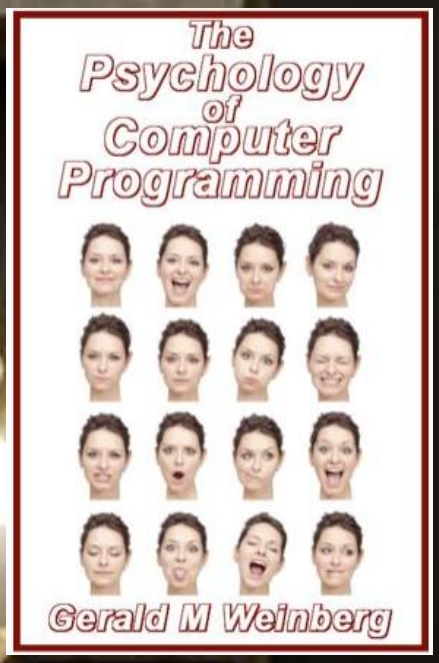
For more than 50 years, he has worked on transforming software organizations. He is author or co-author of many articles and books, including The Psychology of Computer Programming. His books cover all phases of the software life-cycle. They include Exploring Requirements, Rethinking Systems Analysis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



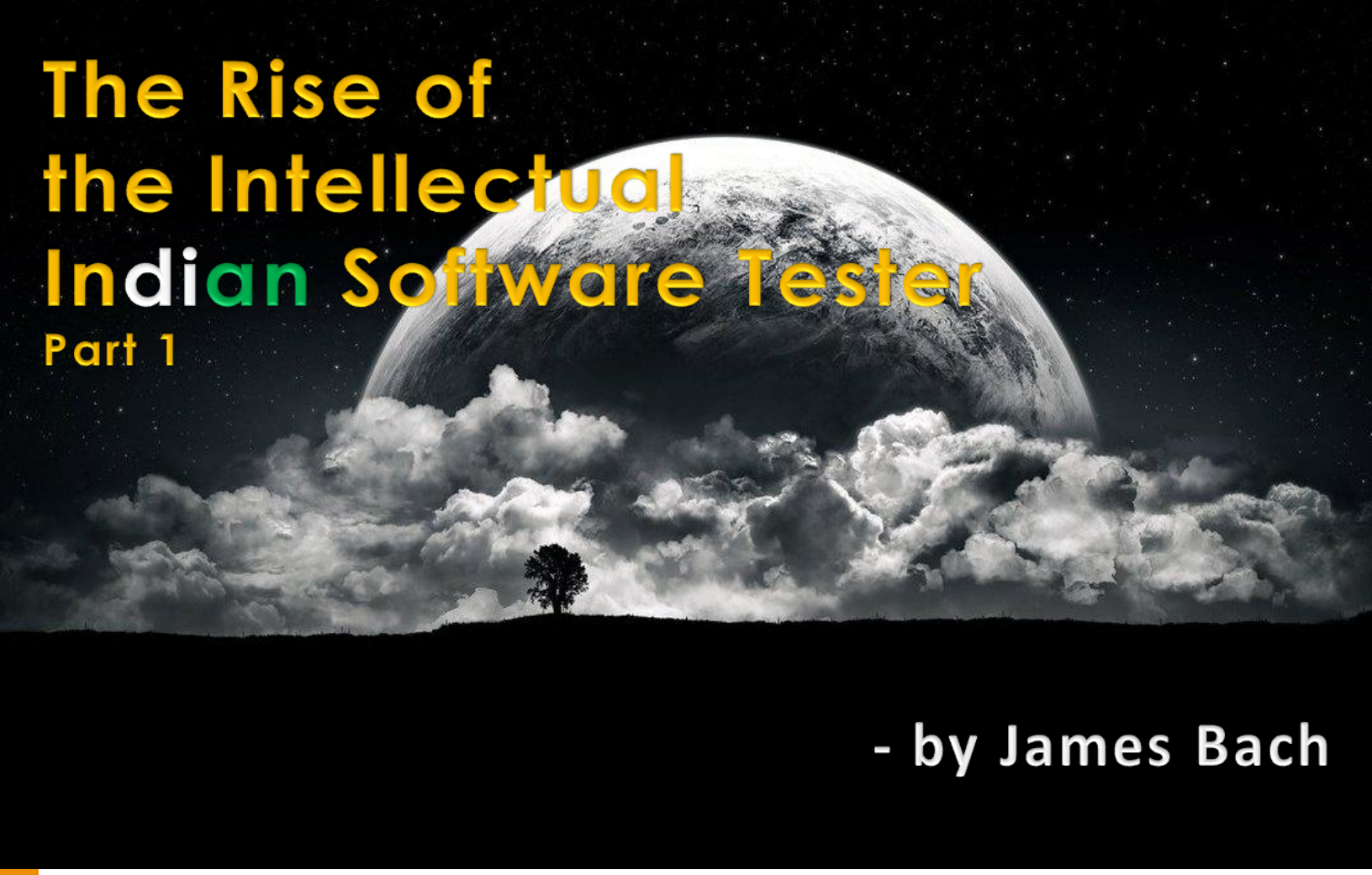
TTWT Rating: ★★★★★

"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



To be a tester is to be a thinker. Testing is not just pressing keys and looking at the screen. Testing is pondering and learning and interacting with a product for the purpose of discovering important and hidden truths about it. Testers are troublefinders.

Strictly speaking, this process cannot be scripted. A scripted process is one that is determined in advance of its execution. One drawback of a fully scripted test process is that it can only find problems that were *specifically and precisely anticipated* at the time the script was produced. But when we test we need to find unanticipated problems, too. We want to find *all* the problems that matter. Another reason testing cannot be scripted is that human curiosity, learning, and confusion—all of which are crucial to a test process—cannot be reduced to an algorithm. Complex problem solving, in general, involves substantial tacit as well as explicit skill.

This brings me to my topic: the Indian testing industry. It is commonly believed in the West that Indian testers cannot work without scripts; that they cannot think in rich and deep ways while testing. Meanwhile, in the last twenty years, in the West, a small but passionate and growing community of testers has been engaged in an intellectual testing. (This is not the "agile testing" movement. I'm referring to the community that calls itself the Context-Driven School. We focus on skills that allow us to operate effectively in any context.)

Intellectual testers don't need notes pinned to their sleeves in order to take action. We design and re-design our own tests, as we go. We are systems thinkers, creative thinkers, and critical thinkers. This is not exactly new, but we brought a new element to it: *systematic* and *collaborative* skill development.

For the first years of this revolution, India played no role in the Context-Driven movement. It was a dark continent. That has now changed.

My First Visit

My introduction to India left me with mixed feelings. I visited Bangalore the first time, nine years ago, at the behest of two of my clients. Both large American companies, they had outsourced some of their testing, but were disappointed with the result. The way my clients described their Indian vendors: they weren't testers at all. They were, at best, script jockeys. This is what my community calls "factory school" testing. (My community calls itself the Context-Driven School. We focus on skills that allow us to operate effectively in any context.)

Check around the Internet. Talk to test managers. This is a common story. You'll hear it all over: "Upper management forced us to outsource to India, and *those testers suck*. (eye roll and sigh.) Oh well, there's nothing to be done."

And so I went to India. But what I found surprised me. This is what I wrote in 2003, just after I returned:

> *I have met quite a few such sharp testers in the U.S. and UK. But, in India I expected to find polite, silent students. I expected that they would be intelligent, but timid about engaging mind-to-mind with me in class, especially when what I'm teaching flies in the face of most traditional advice about testing that they have read and heard.*
>
> *What I found instead were testers who quickly warmed to the challenges I made. They did speak up. They didn't challenge me with the same intellectual swagger typical of testers in, say, England, but they responded to my questions and found novel answers to problems I posed them. In several of the exercises, solutions were proposed that I had not heard from anyone else since I started teaching in 1995.*
>
> *Like most testers, they have not yet developed their potential. But I no longer believe there are important cultural obstacles to hold them back. So, don't be surprised if in a few years the Indians start getting the reputation as insightful, penetrating testers.*

I admitted I was wrong about Indian testers. My brother Jon discovered the same thing and blogged about it six years later, when he trained an Indian team of his own.

> *They loved exploring, were not shy, talked over each other, even gaggled like kindergarteners eager to show each other as if it was show-and-tell time. It was amazing, and it was as easy as a key being turned in a lock.*

Indian Testers Are Not Stupid... So Why Do They Test Badly?

Imagine someone who owns a sports car with a 350-horsepower engine, and yet pulls it around with an elephant—not because gasoline is expensive, but because he thinks turning on the engine would seem arrogant, presumptuous, and maybe the sound of the motor could disturb the neighborhood.

That's my feeling about Indian testers. I met lots of smart people in my Bangalore testing classes. But I got the impression *they didn't feel it was polite to think like a tester*.

Indian culture is not an anti-intellectual culture. But it is a somewhat hierarchical, family-oriented, *service* culture. Good Indians listen to their parents and do their duty as they see it. This attitude also affects their sense of what is polite to say to the boss (or parent or foreign client as the case may be). They want to say "yes" even when they feel the honest answer would be "no" or "I don't understand." This can give the impression of evasiveness in a technical project. Good testers say unpopular, important truths. Testing is a service occupation, yet a misguided sense of service—not wanting to annoy the boss—can ruin the process.

That's one reason for the trouble. Here are some others:

1. English is not the first language of many people in India, making communication inherently difficult in an industry dominated by English speakers.

2. Outsourcing is inherently difficult, but since India is such a popular outsourcing destination, the problems of outsourcing become unfairly confused with the problem of being Indian.

3. A culture that routinely employs another culture is likely to begin thinking of itself as more "senior." It is in a position of power and experience, regardless of its actual competence. Therefore, Western companies may be *predisposed* to thinking of their foreign vendors as confused and timid.

4. No one ever taught them how to test. This is not specific to India, of course, but when combined with cognitive biases known as *Actor-Observer Asymmetry* and *Trait Ascription Bias* (I'm sure I don't need to explain these, because you are good at Google, too) it creates the impression that those guys over there, who happen to be Indian and who seem to be incompetent, are *more* incompetent than our own people (even though they aren't) and are incompetent *because* they are Indian (even if Indian heritage has nothing to do with it).

Taken together, I actually felt this was good news for India. Because it's *fixable.*

> Yes, yes, it seemed clear to me that Indian testers, in general, were pretty bad—but that's just like testers everywhere else. And yes Indian culture looked like a mild handicap, and certainly test outsourcing was particularly hard to do well under the best of circumstances.

But even so, I saw a wonderful potential there, once testers in India began to wake themselves up. I figured it would take a few years.

The Coming of Pradeep

Indeed, it took two years, as I reckon, before the transformation began to happen. It came into my life in the form of a particular man.

January 10th, 2006

Dear James,

Surprising why a stranger addressing you as 'dear', well the reason being we both share something in common - Testing. I introduce myself as Pradeep Soundararajan from Bangalore, India who happened to look through www.satisfice.com.

With that quirky intro, Pradeep requested to become my student. I like having students, but my time is limited, so I gave him an assignment that would require him to do a lot of work: I asked him to test a web site, expecting not to hear from him again. Instead, he completed it the next day.

At that time, I was in the middle of a very intense court case, with very little time to spare. It would take Pradeep 98 days, sending me 19 reminders, before I got around analyzing his work. When I did, I was pleased with I found, and very impressed with his determination. I decided to invest in him.

I don't necessarily charge money to teach testing. Instead, I provide coaching and support to people who have the drive to keep sending me reminder emails. Also, I help people who inspire me, and one thing that inspires me is when they turn around and help others. Pradeep became one of those students; he was a catalyst for the new wave of testing enthusiasts, centered in Bangalore.

Very soon after he contacted me he started a blog, partly to share his testing ideas and partly to practice writing in English. His blog was a call to Indian testers to wake up, and some testers heard his call and responded. He held meetings and taught classes. A new, small, Indian testing community was born. Now it seems like there are a lot of Indian tester blogs devoted to fostering testing skills, but Pradeep was the first of those. He deserves a lot of credit for that, and many of the bloggers he inspired have now come to me for coaching, too.

Another factor that helped was the arrival of the Black Box Software Testing online class. Created by Cem Kaner, with the support of some of my materials, it is a demanding multi-week testing learning experience. People all over the world have signed up for it, including many in India. Between Pradeep's preaching, online resources, remote coaching by Context-Driven testing guys like me, and the BBST class, any ambitious tester in India can now get the support he needs to thrive.

India as a Testing Cuisine

My exposure to Pradeep and others such as Ajay Balamurugadas, Meeta Prakash, Shrini Kulkarni, and Parimala Hariprasad, began to change and broaden my perception of Indian culture and its potential role in building great testers. I began to see the possibilities of Indian testing against the backdrop of thousands of years of philosophy and history.

Please bear in mind that my excitement about India is not a comment on any other culture. Just as I like to eat at Japanese and Italian restaurants, I also enjoy Indian food very much. This is not to say that there's anything wrong with traditional American cooking. In the same way, although I am an American tester, and I appreciate the advantages that American culture brings to the test process, I also am excited about how I can be a better tester by learning about India.

I have dabbled in the study of India for years. Only recently have my studies become focused. I'm interested in the relationship between "thinking like an Indian" and "thinking like a tester." I'd love to see a few Indian testing heroes stand up and do this study better than I ever could, but to get you started, here are some of the elements that can inform excellence of Indian testing:

- **India is a plural society that knows how to adapt.**

  Where else can we find so many people, with so many different religions and sects, living so close together with so little violence? Nowhere. Growing up amidst such diversity may help Indians in technical life, too. Testing requires us to bring together competing ideas and interests.

- **India is especially patient in the face of chaos.**

  To a Western eye, Indian cities are an utter mess; a tsunami of dysfunction. But somehow they seem to work, and even thrive. What sort of mind must be required to tolerate living there? Imagine that same mind encountering chaos in a test project. It may be easier for Indian testers to remain calm through the long parade of outrageous bugs.

- **India understands service, loyalty, and reverence.**

  India is a family and clan-oriented culture. They are generally more comfortable with hierarchy than we are in the west. Whereas this may be a drawback for independent creative thinking, it may a boon when it comes to daily motivation and reliability—as long as the needs and tasks of testing can be framed in terms of service.

- **India has its own tradition of Epistemology.**

  Epistemology is the branch of philosophy concerned with how we know what we think we know. One of the great concerns in Hindu and Buddhist tradition is distinguishing reality from illusion. So, it should not surprise us that India has an ancient tradition of logical, skeptical, and scientific thought that actually pre-dates the Greeks. Most Indians don't study

it and don't even know about it, but it's there as a source strength for those who do, because testing is nothing more than applied Epistemology.

- **India has a rich tradition of heuristic learning.**

  Treasures of Indian literature include the Mahabharata (and Bhagavad Gita), Thirukural, Arthashastra, Panchatantra, and the Buddhist and Nyaya Sutras. Something all of these have in common is the practice of teaching through the consideration of opposing ideas and outright paradoxes. This fosters what I call heuristic learning, which develops the ability to make complex judgments where no clear right answer exists. Testing requires that sort of thinking, because of the impossibility of complete testing.

- **India understands that excellence is achieved through struggle.**

  One of the differences between India and America is that here in America the culture expects instant gratification. Both in the everyday social order and the spiritual literature of India, however, great outcomes are expected to take time and work. This is important because it requires a long, daily struggle, and much experience, to develop deep testing skills.

I would like to thank Michael Bolton and Mary Alton for their help with art and editing. *In part 2 of this article, I will describe my return to India after nine years. I taught at Intel, and Barclays, re-visited Mindtree, and spent several days with Pradeep and the crew at Moolya. The star of Indian testing is rising.*

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

Looking for RIGHT job in Software Testing?

visit Qualityjobsportal.com

after all, it's your career we are talking about !

Do YOU have IT in you what it takes to be GOOD Testing Coach?

We are looking for skilled ONLINE TRAINERS for Manual Testing, Database Testing and Automation Tools like Selenium, QTP, Loadrunner, Quality Center, JMeter and SoapUI.

TEA-TIME WITH TESTERS in association with QUALITY LEARNING is offering you this unique opportunity.

If you think that YOU are the PLAYER then send your profiles to trainers@qualitylearning.in .

Click here to know more

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

Hi Guiding Star,

The article "The Testing Dead - The Zombie Menagerie" episode 1 in the Oct 2012 issue stated that it is a mistake for QA Engineers to position themselves as the gatekeepers of the software release decision and are "confused" if they take the blame when some bugs go into production. Those are fine statements but in reality testing is typically the last step in the development process and other teams do look to the testing team for final sign off. And typically when some bugs do go into production it's because the testing team didn't cover those scenarios in there testing.

I am interested in knowing ways the testing team can solve these two dilemmas and not just mere rhetoric. What specific advice can you provide for the testing teams to not become gatekeepers and not get blamed for production issues?

Thank you for taking the time to answer.

- Chinh Q. Tran

"The article "The Testing Dead - The Zombie Menagerie" episode 1 in the Oct 2012 issue stated that it is a mistake for QA Engineers to position themselves as the gatekeepers of the software release decision and are "confused" if they take the blame when some bugs go into production."

Not quite.

I said that it is a mistake for Software Testers to position themselves as the (http://trenchescomic.com/tales/post/13028) gatekeepers of quality and the release decision. I also said that 'The Confused' are people who believe they are doing quality assurance but are in fact software testers. They tend to enjoy grandiose titles such as 'QA Engineer' despite not doing QA and not being an engineer. Quality Assurance and Software testing are related disciplines, but they are (http://www.testingeducation.org/a/TheOngoingRevolution.pdf) not synonymous (see page 6 and beyond).

This comes back to what a software tester's role is at a fundamental level. The role of a software tester is to reveal information about the product and its artifacts to people that matter. Their role is to inform their audience (be it project manager, team leader, programmers, analysts, site producer etc) about what they know, what they suspect, what they have done, what they have not been able to do. They need to do it in such a way that it is (http://testjutsu.com/2011/09/framing-your-tests-framing-your-audience) meaningful to that audience.

Hunting for bugs and finding information about product stability is frequently part of a tester's role. It does not follow however, that their job is to declare the product bug free, nor is it to fall on their sword should bugs make it to production. The release decision involves more than an understanding of the technical aspects of the product or project. There are business aspects that the software tester is generally not privy to. I think it is folly for a tester to assume they know enough to put themselves in this position.

"in reality testing is typically the last step in the development process and other teams do look to the testing team for final sign off."

I have worked in organizations where this was generally accepted to be the case. My first steps have always been to help disabuse my peers of these misguided notions. Testing begins when testers can help uncover information that is useful to their audience. Analyzing and questioning designs is testing. Challenging assumptions around behaviour and usability is testing. Comparing design wireframes to previous versions or accepted industry norms is testing. Sitting with a developer as they code and asking questions is testing.

Testing is not a finite process at the end of which is a green light to release. It is a continuous process of uncovering information, challenging assumptions, learning, reporting up until (http://www.developsense.com/blog/2009/09/when-do-we-stop-test/) the point where testing is done. The release decision should be left to the people who are paid to make it - typically the project manager or the person(s) they report to. Your job as a tester is to inform them about risk from a technical standpoint so they can add that to their knowledge of the business needs in order to make an educated decision. By all means make recommendations about the stability of the product based on what you know about it, but allowing them to abdicate responsibility for the release decision is fraught with peril.

"and typically when some bugs do go into production it's because the testing team didn't cover those scenarios in there(sic) testing."

Typically? I'm not so sure. I suggest that your statement is a typical conclusion that some people in our industry arrive at and I think that is precisely because testers who are confused about their role position themselves as quality gatekeepers, or are ignorant about their role and allow themselves to be positioned there. 'You said it was good to go. It obviously isn't good to go, so you are at fault'. Sound familiar?

I'm certainly not saying that the testing team should shirk responsibility. If there are issues that make it to production that a tester (or testing team) could and should have found during their work, then they need to put their hand up for it. Is it a possibility that the testers missed something? Sure. There are an infinite number of alternate possibilities also. It's possible that the initial design was flawed and not corrected. It's also possible that differences between testing environments and production meant some issues were not revealed until after release. It could be that a crash only occurs with a certain set of key combinations or that a compiler issue causes a crash after an action is performed for the 256th time. Software quality is everyone's responsibility. Part of your job might be to help jog people's thinking about what could go wrong. It doesn't mean you take responsibility for doing their thinking for them also.

What specific advice can I give you not to become a quality gatekeeper and not get blamed?

Being aware that this is a stupid place for a tester to be is a good start. Helping your peers become aware of this is better.

If you are in a situation where people are looking to you as a software tester to make the release decision, then you probably have your work cut out for you. It's likely there's a culture that exists in that company that sets the expectation and it will be up to you to help change it. There's no magic set of steps you can take to make this happen.

My first suggestion would be - if you are a software tester, stop calling yourself anything with the words 'Quality Assurance' in it, and call yourself a 'Software Tester'. It seems a small thing, but it is powerful. It will help you have some of the difficult conversations you're going to need to have, and it will help your peers better understand your role.

It will be a process of building relationships and education to varying degrees. It may well be a slow process. It will likely make a lot of people uncomfortable (including you). You will encounter resistance, possibly hostility, possibly even accusations of shirking your responsibility. You need your peers to understand that your role is to reveal and share useful information. Not to make promises that the software is bullet proof. You will need to help them understand that software quality is the responsibility of everyone working on it. That goes for design as much as it does implementation. As a tester, you are not making changes. You're informing other people about issues and they are making changes. You can perhaps influence quality. There's nothing you can do as a tester to assure it.

Get better with your (http://testjutsu.com/2011/10/pull-reporting-push-reporting/) test reporting. Learn how to add value beyond 'this passed, this failed'. Understand what information your audience needs and give it to them in a way that is useful to them. If you can be seen as a skilled knowledge worker and not a grudgingly accepted cog in the software development process, then you may start changing some minds. Ultimately, it's up to you to determine how best to do this in your organization.

- Ben Kelly

Tea-time with Testers

Book Review

I just finished reading Gojko Adzic's latest book Impact Mapping and here are some impressions this book left on me.

Things that I liked:

- Look, Feel and Style

  Being a magazine designer myself, I always try to keep the design easy to read and appealing enough to keep the reader engaged with the idea. 'Impact Mapping' won my heart with its simple yet appealing design. Appropriate use of cartoons has made it even more interesting.

- Problem Analysis and Explanation

  Thing I liked most in this book is 'excellent problem analysis and the way it has been explained to readers'. Gojko's analysis around failures in software development reveals his rich experience and study on this subject.

- Narration

  It's not just enough to know the solution to any problem but it's equally important to be able to explain/present it in a manner that others will understand and give it a thought.

  I liked the way Gojko has explained his idea of Impact Maps. Each topic is well-connected with the other one and that helped me to stay away from confusion.

- The Concept

I won't tell you what Impact Mapping is all about (because Gojko has done that job well and it would be great experience to learn it via his book itself). But yes, if you are fond of mind-mapping and if visual techniques attract you then you'll certainly like the idea and you'll like it even more once you try your hands out (I feel).

I liked the concept, especially because it's a powerful tool to find out whether you are doing right things at right time and with right set of resources, or not. It can also help to address/identify the 'Problem of many', if you get it right.

[Hint - Gojko Adzic (with David Evans) had written an article 'Visualising Quality' in **July'11** issue of Tea-time with Testers. I feel that part of that article can give you slight idea about Impact Mapping :-)]

Things that I would have loved to see:

- It's not necessary but IMO, some additional real life examples (from simple to complex) could have helped readers to re-check with their interpretation of the idea. Or may be some 'Try this Out' kind of exercise (wherever applicable) could have been an added advantage.

- I am passionate tester and I would have loved to get some special tips for testers on how testers can make effective use the concept. Especially, where testing teams operate separately and not as a part of development team.

Conclusion:

Professionals with experience of (or inclination towards) Agile development methodology are surely going to like this book because it provides simple yet powerful solution to majority of problems that Agile teams often face.

If your organization follows that iterative approach or if it is planning to adopt Agile development methodology then this book is a must read.

Hope you find this review helpful.

- Lalitkumar Bhamare

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

Balancing Time with 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.

Based upon the feedback received on them, I decided to write a couple more articles to address a few more areas. We all know that we will never have the amount of time to test that we want. Now your company adds cross-browser testing, but may not increase your testing time by much. Therefore, one area I would like to touch upon is how to minimize some of your testing across browsers. For the final part of this series, I will share a couple of mind maps that I find useful in organizing my testing.

What browsers do you test?

Most likely you will be provided with a list of browsers that need to be supported. Our first thought is we need to test all of them. But do you? Some browsers (i.e., Chrome and Safari) use the same webkit that may eliminate testing certain browsers or allow you to sample the testing across them. Have a conversation with your technical employees to review the supported browsers and any similarities between them. Understand any potential risks to reducing testing. From this information, determine if you can eliminate testing a browser.

Review this decision with the department who is in charge of the cross-browser project to gain buy-in on any reduction to testing approaches. You do not want to surprise them at the end by telling them you did not test a browser. If they have concerns not supported by the technical employees, propose a compromise. Test one of the browsers and test any identified bugs in the second browser and perform a level of sampling. For example, test some of the critical areas that passed testing in the first browser to ensure it works in the second. Basically you want to gather sufficient testing evidence to support eliminating testing a browser.

Remember, your level of cross-browser testing should be based upon risk and not testing everything in all browsers. If you can eliminate or sample the testing across similar browsers you can save a tremendous amount of time.

IE

IE is still a popular browser though it is getting serious competition from Chrome. Often we are asked to test several versions of IE. Again this can become very time-consuming. A good question to ask your team: "what is the risk if we do not test everything on all supported versions of IE?"

• What intelligence could we gather to help answer this question through testing?

• What conversations could you have with the developers?

• If there are risks, can you identify the specific areas of your product at risk?

• Can the testers have different versions of IE? For example, if the testers were all on IE7, only have some of them upgrade to IE8 and IE9. This allows testing across 3-versions with the understanding that new functionality and bug fixes will not be tested across all versions unless a specific risk has been identified.

Reducing testing time across browsers

It is important to understand how your product responds to the difference between the browsers. In some cases you may only perform a smoke test that you can open the functionality across browsers with a simple test or two. In other situations you will need to perform more testing because the risk between the code and browser is greater. In these cases, talk with the developers to understand the risks. Then determine if one of the following strategies is feasible to reduce testing time:

• Split your testing activities across supported browsers. A spreadsheet may help you manage what features / tests are being performed on each browser.

• Conduct your testing on your main browser supported. Identify the core or critical tests that should be performed across the other browsers. Identify these tests based upon conversations with the developer and the history of problems encountered across browsers. Also consider the critical functionality to the clients to ensure risks are mitigated.

• Do some browsers require less testing because a small percentage of your clients are using the browser? Can you test those browsers last in case you run out of time? Or define a smaller subset of tests?

• Understand if the browsers all fail in the same manner. Are you encountering problems in one browser that you are not seeing in the other browsers? How can you use this information to better target testing?

• Are the developers writing code to identify which browser to fix cross-browser problems? How does that change your testing approach?

Chrome and FireFox

Chrome and FireFox do not support multiple versions of their browsers eliminating the need to test across versions. When an update is available for Chrome, a little arrow icon is displayed on the toolbar. There is an update option and you will need to restart your browser. Firefox automatically updates its browsers. They are installed when you restart Firefox. If that does not happen, you can manually update it by going to the Help menu and selecting "About Firefox". For both browsers you can learn more about how they provide updates by going to their support page.

Product Vulnerability

When you perform your first cross-browser testing, gather as much intelligence on the types of problems found and the associated browser(s). Examples of problems include information shifting on the screen; font size too large or small; extracts not created; and information not saved correctly. It can be helpful to store this information in a spreadsheet or database. I like to categorize these problems to better understand what percentage of the problems are look and feel, data, and functionality. From this information you can identify the minimum subset of testing you should perform based upon the coding risk such as JavaScript and type of risk such as look and feel.

You may have to test over several releases that go to production to gather your initial intelligence. This is an ongoing process because over time your recommendations can change. However, after a few releases you may need to gather less intelligence unless a new coding technique is introduced such as JavaScript or adding new browsers. When that happens, perform a gap analysis to understand any potential risks to determine testing approach.

Product Vulnerability Spreadsheet

I like to use a spreadsheet to track the intelligence we are gathering on cross-browser testing. But just as important, I perform hands on testing across the browsers and discuss with the testers what they are witnessing in testing. Be careful to not make all your decisions based upon the spreadsheet or database. I often get a better understanding of the risks we are facing through discussions rather than spreadsheets and numbers.

Data I tend to collect: browser and version; test build; date; tester; category; product area (i.e., report name, module name); priority of the fix; and brief description of the problem. You may consider adding other columns such as coding technique used (i.e., JavaScript, HTML).

Opportunity Costs

Whenever identifying testing strategies and approaches, there is always an opportunity cost of your decision. If you extensively test all browsers when it is not warranted what was the opportunity cost? What testing did you reduce or not perform that may have identified new risks or important bugs?

Security

One area I did not discuss in my series was security and cross-browser testing. Ajay Balamurugadas brought this to my attention and I am glad he did since it is important. Many products allow you to set security or permissions levels for a user. He may have permissions to perform all functions (i.e., create, edit, load data) while another user may only be able to view information or interact with a subset of the product. Perform a series of tests with different permission levels to see if you encounter any problems across browsers. Identify a sample of your bugs and retest with different permissions to see if you get the same results.

Since time is always short when testing, gathering initial intelligence related to security and permissions will help identify future testing.

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

Back To Index

A click here will take you there ☺

It's time to turn the kill switch on

- a testing-only approach to software quality

By Lev Lesokhin

When a high-frequency trading (HFT) glitch causes a crash on Wall Street, the results are felt immediately. Investors' faith in the market weakens, pink slips are handed out, and, during it all, one unlucky company helplessly watches its value plummet. It's like a runaway train on Wall Street, and no one is quite sure how to stop it.

The SEC recommended "kill switches" be placed on the trading networks to immediately shut down any rogue algorithm or trader. But this is a jury-rigged attempt to fix the problem without examining the underlying causes of why the crashes are happening in the first place.

To actually fix the problems we're having on Wall Street, financial firms will need to maintain a high level of software quality while their trading systems are being built. This explains why these crashes are becoming more and more common -- we're regulating the traders, and ignoring the software.

The architectural diversity that gives modern business applications their unique power and flexibility comes at a cost of staggering complexity. Quite simply, the complexity of modern business applications exceeds the capability of any single individual or team to understand all of the potential interactions among the components, let alone the languages and technologies deployed. Organizations are now faced with the devastating impact of architecturally complex engineering violations.

An architecturally complex violation is a structural flaw involving interactions among multiple components that might reside in different application layers. Whatever component causes the violation is typically in or around an architectural hot spot.

So, how can firms and other organizations prevent architecturally complex defects in their software? This article will answer the most basic questions about these defects, as well as offer advice to prevent them.

- **Why do architecturally complex violations take more effort to fix?**

They are multi-component and therefore require a lot more files to fix than a code-level violation. Reported data indicates that, frequently, as many as 20 different modifications to files are required to remediate a single architecturally complex defect.

- **Why are architecturally complex violations more costly to fix?**

These defects are more expensive to fix because they involve interactions between multiple tiers of the application often written in different languages and hosted on different platforms. These violations require much more involvement and coordination across teams to ensure that the fix is resolved system-wide.

- **Why are architecturally complex violations worse as they cross phases?**

Since complex violations are more likely to persist into operations, they are more likely to cause operational problems than the single component violations that tend to get caught earlier. And they take more testing to detect in the first place.

- **What is the map of decay, and how is it used?**

A map of the most frequently fixed relationships among architectural hotspots reveals the architecture of decay. But it also presents a roadmap to guide high-value remediation and the greatest opportunities to restore the structural health of an application. Big problems are often the result of several interacting weaknesses in the code, none of which caused the problem by itself. Preventing application-level defects requires analysis of all the interactions between components of heterogeneous technologies. Reliably detecting software quality problems requires an analysis of each application component in the context of the entire application as a whole – an evaluation of application quality rather than component quality.

Maintaining high software quality using the tips above seems like the best way to eliminate the growing problem that high-frequency trading has brought to the market. But that's not something most firms want to hear -- and they're not going to ask for more regulation anytime soon either. They're concerned about getting their programs to run faster and trade smarter … and that's about it.

And that's unfortunate, because crashes like Knight Capital and NYSE will continue to become more common until Wall Street realizes that it is no longer in control of the monster trading network it has created. And unless we want rogue algorithms bouncing stock prices around like a schooner in a hurricane, we need to ensure architectural quality in our financial systems before it's too late.

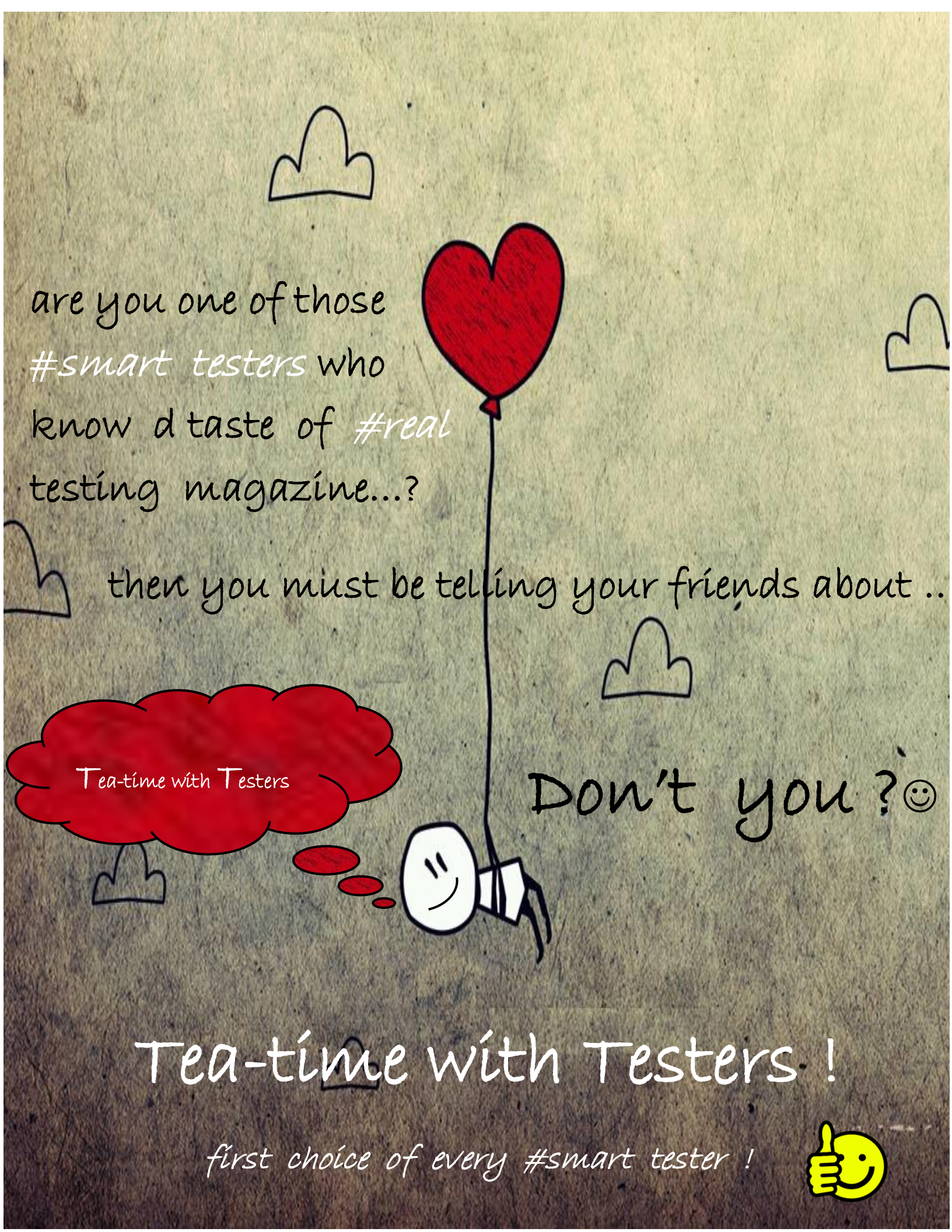
Lev Lesokhin

Executive Vice President, Strategy and Market Development

Lev Lesokhin is responsible for CAST's market development, strategy, thought leadership, and product marketing worldwide. He has a long career in IT and apps dev, a passion for making customers successful, building the ecosystem, and advancing the state of the art in business technology. Lev comes to CAST from SAP, where he was Director, Global SME Marketing. Prior to SAP, Lev was at the Corporate Executive Board as one of the leaders of the Applications Executive Council, where he worked with the heads of applications organizations at Fortune 1000 companies to identify best management practices.

http://www.castsoftware.com/

Back To Index


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**

Master Test Plan – the strategic side of testing

There are many software organizations where the term QA refers to a group of testers who randomly receive builds from the developers and proceed to "play" with the system in order fish out the bugs. In some cases these engineers use some informal flows or scripts to base their tests but these documents are not even close to a testing strategy or test plan of any sort.

We all agree that one of the main objectives of the QA team is to rid the product of the most important and disruptive bugs, but doing it completely Ad-Hoc and without a good preparation is the most ineffective and inefficient way to go about this process.

The common testing project is composed of 4 phases:

a. Planning

b. Preparation

c. Execution

d. Post-analysis

In this post I will focus on the planning stage, since I believe this is the most important phase and the one that we take for granted the most.

So now we can start with 2 big questions:

- What do we need to plan?

AND

- How do we go about planning it?

The answer to both questions is the MTP or Master Test Plan (also known as Software Test Plan, Testing Strategy, etc). The MTP is both a document and a process; by this I mean that at the end of the day you will have a document you can look at and admire (you may even hang it on the wall!), but not less important is the process you need to follow to create and communicate all the aspects that conform the document with all its sections.

What makes a good MTP?

Each company has different needs and thus each will require a different MTP template. The important thing is to understand that this document will represent your Scope of Work (SOW) for the specific project. It should be the place where you and your external stakeholders (Product Management, Development, Support, etc) turn to in order to understand what your team is testing and how are they approaching each testing task.

To look at it in a simple way, imagine your company decides to outsource all its testing tasks to an external group (your group) and you need to put together a contract explaining what your team will do and what will it need in order to do it. Like all contracts, the idea is to review all the details and agree on them before signing the deal (or starting the project).

Following are the usual sections I include in my MTPs (remember to take them only as a suggestion and to modify them based on your needs!):

1. Objectives of the testing process:

The objectives of the testing process depend on the nature of the development project. Examples of testing objectives are: new feature validation, additional configuration certification, translation validations, installation and/or upgrade testing, etc.

Just make sure you don't write trivial stuff like: to find all the bugs in the system or to assure we release a quality product.

This section is to communicate what YOU will be thinking when planning and running your tests.

2. Testing scope:

For many people the testing scope is the heart of the Master Test Plan. It describes the things you will focus in each of the application areas and/or features to test. I tend to make this section a nested list, and for each item I describe:

- The main aspects to tests

- The product risks or potential bugs I foresee

- Concrete faults or main scenarios to validate

- Assumptions or requirements (documented API, stable GUI, etc)

- And any other aspects worth mentioning regarding the specific area under test.

This is the place where you provide your stakeholders with the information about what will you be testing on each section of the product, and here is also where you should look for comments and suggestions from developers, product architects, support engineers, fellow testers, etc.

Some MTPs go the extra mile and provide a list of areas and features that are Out of the Scope of the testing process.

3. Testing configuration matrix:

Your application surely needs to support a defined number of configurations and platforms, here is where you should list these configurations together with the testing matrix you will run in order to validate it.

Keep in mind that by configurations we may refer to different things for different projects; on one project it may be Operating Systems and Browsers, while on another project it is additional product-components and specific versions required by your product to function correctly.

In any case, by configuration we mean the environmental (and thus external) parameters required by our system to work.

In the cases when the list of officially supported configurations is more extensive than the systems you plan to test you should provide 2 separate lists:

(a) The list of theoretically supported configurations, as specified by your customer or product management team.

(b) The list of actual configurations you will test in order to achieve the above level of support, together with the distribution or percentage of tests that will be run on each.

In order to do this I suggest a presentation format and planning method similar to what I described in my last post.

4. Required Hardware / Software

Based on the testing scope and the required configurations you need to create a list of all the hardware and software resources you will need to complete your tests.

In addition to special machinery and/or licenses, this is the place to ask for specific stubs or simulators you may require during your tests.

If you plan to use automation of any sort include the number of software licenses as well as the amount of virtual users you will need for load and performance testing.

5. Testing preparations

By now it should be clear what you want to test, now you need to understand what preparations to do in order to test it.

For this point should make a high level review of your test plan inventory and compare it to your testing scope. You should end with tree lists of tests:

(1) Tests that are ready to be run as is

(2) Tests that need to be reviewed and/or updated to match the changes to the functionality

(3) Tests you need to write from scratch

For each list assign the amount of time you will require to work on the tests; you can also include the information and/or help you will need from other teams.

If you also need time to prepare testing environment or create testing data this is the section where to add this additional preparation costs

6. Testing schedule:

(The favorite section of all our Project Managers!)

Our operations are usually divided into stages and cycles. For example a project may have a preparation stage, an execution stage composed of 3 to 5 testing cycles, and a final product release stage/cycle.

For each one we should provide at least the following information:

(a) Expected time lines

(b) Entry and exit criteria

(c) Testing scope & objectives

(d) Testing resources (peopleware, software and hardware)

and any additional information pertaining to the specific cycle.

7. Testers & schedules

Following on the project side of the MTP you should list all your testers and the dates when they will be available for your project. List holidays, vacations, training and any other activities that may have an impact on the availability of a resource.

If part of your tester-resources will be new make sure to account for the training and ramp-up time they will require at the start of their work.

8. Risks

Like every project you should list the risk you may encounter. Examples of risks are difficulties in recruiting resources, instability of the product that may delay your schedule, high attrition rates, etc.

Each risk should include the following information:

(a) Person in charge of the risk

(b) Severity and likelihood of the risk materializing

(c) Dates of relevancy when the risk may materialize

(d) Consequence of the risk materializing

(e) Prevention & contingency plans

9. References & attachments

Add links to any additional documents and/or information referent to the project.

Add also a list of all the contact persons as well as their areas of responsibility.

A final word of advice - Some companies and processes are not ready to handle a full blown MTP, specially start-ups and/or companies working under less structured development process (e.g. Agile Development).

If you work on one of these companies it doesn't mean that you can or should work without a strategic QA plan, it simply means that you need to mold it in order for the plan to meet your company culture and way of life.

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

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

Year++. Stay young, Have fun.

As we step into the New Year, it is a wonderful time to look forward to new learning, new roles, and new experiences. It is that time of the year when we make interesting resolutions. Resolutions to learn something new, to do new things, to contribute to a better world and so on.

Let us mull over this…

Learning is about acquiring new competencies, knowledge to be able to do things. Knowledge does give us a sense of power, makes us feel nice. But is mere knowledge good enough? Nah - It is only when we apply this knowledge to solve a problem successfully, do we feel a sense of true power. We look forward to application of this knowledge multiple times and only when we fail (and subsequently learn) do we feel the absolute power. This is when we become skilled.

So the FIRST transformation - KNOWLEDGE to SKILL. From knowing how-to-do to actually solving problems successfully. Note successful problem solving requires not only how-to-do, but also how-not-to-do. Don't be afraid to fail.

So as we step into the New Year, it is wonderful to look forward to new learning, successful applications and also failures. Hence there is no need for fear/tension to succeed only; this allows us to learn well.

Now onto the next stage ... Is it good enough to possess the skill to solve problems when asked to? It is indeed nice to be called in to solve problems. It is like 'giving something' when asked for. Guess what, it feels nicer to take ownership, take responsibility rather that just be a passive bystander who can 'give' when asked for. This is absolute power, the power to change, using your skills. So take responsibility, take ownership in anything you do.

The second transformation therefore is SKILL to RESPONSIBILITY, from ability-to-do to own-the-problem. So are you thinking of what you are going to take charge of, in the New Year?

On taking ownership to solve problems, we become good at problem decomposition, activity planning, and executing the activities using the acquired skills. The focus becomes the activities and the successful completion of each one of them. But is successful completion of activities good enough? This requires us to go the next stage. From successful completion of activities to delivering outcomes i.e. delivering business value.

It is the shift from the activities to the recipient of the activities. It is about ensuring the recipient(s), (i.e. end users) benefit from the solution. Delivering value to customers/end-users, rather than just successfully solving the problem. Do you what value you are delivering to your customers now? What value do you intend to deliver to your customer, your company, your team this year?

This the third transformation from ACTIVITIES to OUTCOMES, from doing good work to delivering value to your customers, company and the team. It is when we shift from the 'doing' to the 'recipient' from the 'inanimate-activity' to the 'personal-outcome'.

Continuing the same train of questioning, Is delivering value good enough? Let us think this through.. As we continue to perform activities that deliver value, we do become busy. That is when monotony sets in, tiredness creeps in, and work gets heavier and boring. And it is about getting stuff done, no more fun. Now it is high time for the next stage of transformation. The shift from doing work to having fun. The shift from 'recipient' to the 'doer'.

In addition to seeing the larger picture of activities from the recipient's view, when we immerse in the activities with the focus on the doer, time stops, you are in flow and it is pure joy. That is what work should be - Joyful.

This is the final transformation from WORK to FUN. From accomplishing for others to accomplishing for yourself. This is when you are do things for the sheer joy of doing and guess what, the outcomes not only deliver value, and they are beautiful. This is what each one of us should wish in the New Year, having fun, being joyful and value delivery happens. This is about staying in the present, being relaxed, enjoying every moment, being immersive and outcomes are magical. This is when you become young at heart, and experience the joy of a child.

I would love to share my story of cycling with you now. About a year ago I started cycling, with the focus on long distance rides. Initially it was about building competence on ride techniques, strength building, climbs, hydration, posture and others with focus on building endurance skills. With time and failures I became skilled in doing 100 kms rides with ease. Now I wanted to graduate to doing rides of 300kms and beyond. That is when I discovered that it was necessary to relax and be in the moment as just focusing on time and distance outcomes were weighing me down. Just focusing on the wheel on the front, every pedal rotation, enabled me to be in the moment, and 300 kms was fun to ride, feeling energetic even after completion of the ride. I am looking to doing 500+ long rides this year and look forward to enjoying and completing these this year. Transformation from Stage 1 to 4.

Summarising ...

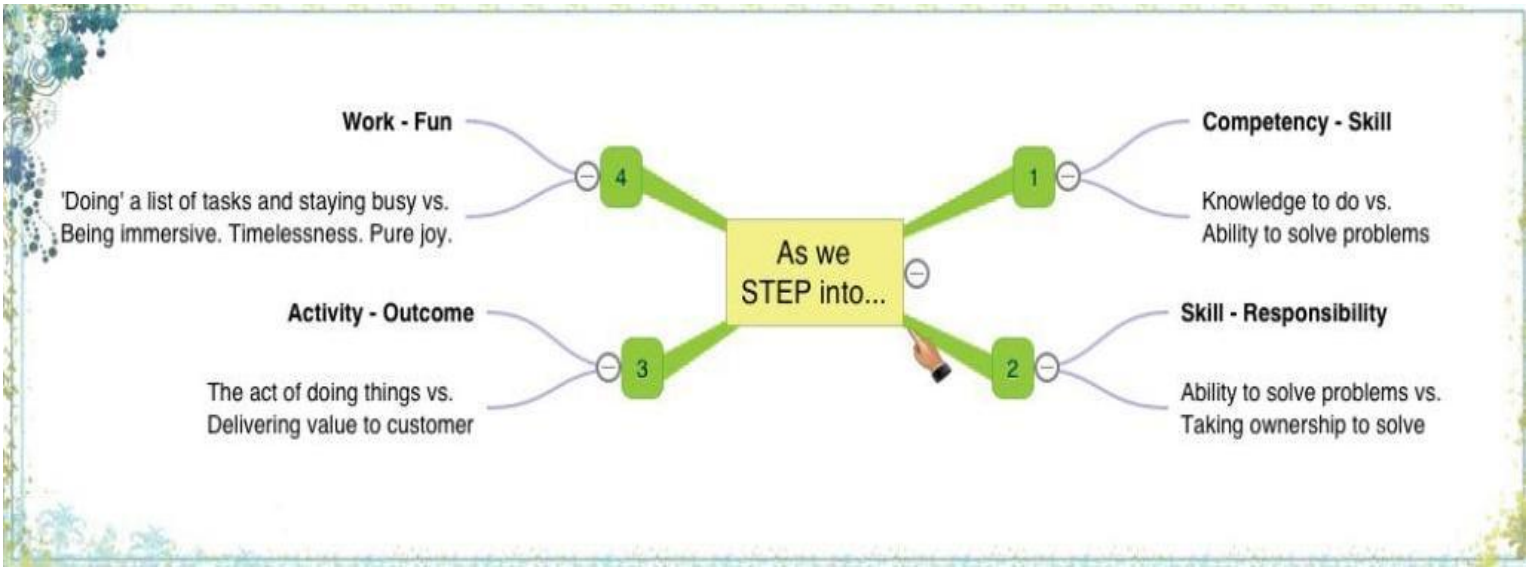
Competence -> Skill -> Ownership ->Activities -> Value -> Fun.

As you step into the new year, ask yourself what you want to know, to be skilled in, be responsible for, what value to deliver, and more importantly have fun.

Have a great 2013. Become knowledgeable, skilled, enrich others and enjoy.

Grow older - year++, Stay young & Have fun.

God bless you.



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**

Back To Index

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

Testing PUZZLES by Sebi

Claim your Smart Tester of The Month Award. Send us an answer for the Puzzle and Crossword bellow b4 31st Jan. 2013 & grab your Title.

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

“The Magic Number”

A magic number is a number that contains all the consecutive digits from 1 to n, where n is the length of the magic number.

Example 632415 is a magic number because it contains 1,2,3,4,5,6 ("6" is the length).

But 212 is not a magic number because not all the digits 1,2,3 are included in it. (So it’s a permutation in a way).

The question is how many pairs of two magic numbers that have four digits have its product equal with another magic number?

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

Answers for last month's Crossword:

E	M	U	L	A	T	O	R
X		T		U			E
P	Y	L	O	T	C	W	C
E		O		G			O
C	R	A	S	H			V
T		D		C	S	T	E
		U			T		R
	S	I	K	U	L	I	Y

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

Hello T.T.W.T. Team,

I am regular reader of you magazine. From November 2012 magazine I liked the article by "Ben Kelly". I found it very usefully to all of us (testers). I understood many important things from the article e.g. the importance of tester in organisation, responsibility of tester and How to improve Dev-QA relations.

New things that I learned from your magazine, I am trying to implement them at my work.

At time I used to wonder, Why Dev teams underestimate QA?

When I joined my organisation I had fear in my mind because I had no prior experience in this field. But later I decided not to fear and not to be scared of Dev teams. I focused on doing my job well and I am glad that I'm doing it well.

I am writing this because I want to say that every tester within an organisation is important and without testers no software business can run well.

I want to thank *Tea Time with Tester* team for providing great knowledge and platform to share our views.

- **Shivendra Pratap Singh**
Pune, India

in next issue

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-weiphoto.com

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