

Tea-time with Testers

Year 7 | Issue 1

Articles by –

Jerry Weinberg

John Stevenson

T Ashok

Marcin Sikorski

Nir Gallner

Sushma Suresh

Alex Placid

Viktor Slavchev

Joel Montvelisky



Over a Cup of Tea with Zoltán Molnár

CAST2017



WHAT THE HECK DO TESTERS REALLY DO?

CAST 2017 is taking a deep look at the practice of software testing. We want to help testers become so good they cannot be ignored. CAST 2017 focuses on the actual tactical work required to perform excellent, effective and influential testing.

Today, testers are challenged to identify important product risks in turbulent contexts. Business focus follows a fickle consumer market, harshly driven by demanding investors. Organizational frameworks are evolving and changing, often within a product's lifetime. Solution technology frantically advances moving from tiered architectures to micro services among a hyper distributed internet of things.

Emerging practices sometimes shuffle testing activities earlier, or later in the life cycle. Some development approaches may even obscure testing. CAST 2017 will focus on the tactical work testers do in a variety of contexts with different tools and techniques.

JOIN US

FOR OUR 12TH ANNUAL
CONFERENCE AT THE
**GAYLORD OPRYLAND
HOTEL IN NASHVILLE, TN,**

**AUGUST
16 – 18 2017**





TEA-TIME WITH TESTERS

First Indian testing magazine to reach 115 countries in the world !

Created and Published by:

Tea-time with Testers.
B2-101, Atlanta, Wakad Road
Pune-411057
Maharashtra, India.

Editorial and Advertising Enquiries:

- editor@teatimewithtesters.com
- sales@teatimewithtesters.com

Lalit: (+99) 15227220745
Pratik: (+99) 15215673149

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

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

Editorial

An Important Announcement

Dear Reader

Hope things have been great at your end. I would like to thank you for your inquiries about coming editions and also for your patience.

Well, like everyone else the time for me has come, that I slow down a bit and do (out of many) things I have been doing with more focus and attention. Serving testing community by all possible means to the best of my abilities has been my passion and I would continue to do so. However, in order to maintain the standard and create the value out of things per my satisfaction, I have decided to slow down on certain initiatives and Tea-time with Testers is one of them.

What does that mean? Well, nothing as deadly as some might think ☺ It's just that for short period of time, TTWT will be published bi-monthly instead of every month as we have been doing over years. When I feel, I have spent enough time on other projects, TTWT will be back in action every month.

I am sorry if that has disappointed anyone in any way. I promise, this will be just for the short term. And in turn, it's only going to make TTWT even more useful for all of you.

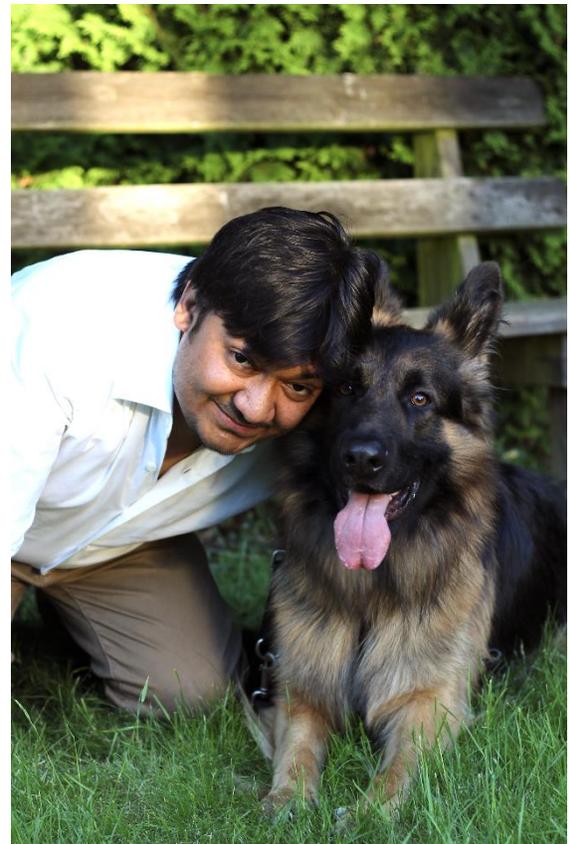
That said, I would take your leave for now. Worry not, we'll keep in touch and you'll hear from us once in a while.

Enjoy the issue and keep those awesome letters coming. That's such a great motivation.

Sincerely Yours,



- **Lalitkumar Bhamare**
editor@teatimewithtesters.com
@Lalitbhamare / @TtimewidTesters



QuickLook



Editorial

What's making News?

Tea & Testing with Jerry Weinberg

Speaking Tester's Mind

Tester's Wildlife - 22

Teach Them All to Code - 25

Testing Skills – part 8 - 30

In the School of Testing

Data Migration with multi-geo Roll Out – 35

Outdated Testing Concepts #3 – 39

Testing is Not A Career – 49

T' Talks

Be in FLOW. Test BRILLIANTLY - 45

Over a Cup of Tea with Zoltan Molnar

Family de Tea-time with Testers



A "Bug Fix" That Could Unlock the Web for Millions Around the World

By [Mike Orcutt](#) – [MIT Technology Review](#) –

Companies that do business online are missing out on billions in annual sales thanks to a bug that is keeping their systems incompatible with Internet domain names made of non-Latin characters. Fixing it could also bring another 17 million people who speak Russian, Chinese, Arabic, Vietnamese, and Indian languages online.

Those are the conclusions of a [new study](#) by an industry-led group sponsored by the International Corporation for Assigned Names and Numbers (ICANN), the organization responsible for maintaining the list of valid Internet domain names. The objective of the so-called [Universal Acceptance Steering Group](#), which includes representatives from a number of Internet companies including Microsoft and GoDaddy, is to encourage software developers and service providers to update how their systems validate the string of characters to the right of the dot in a domain name or e-mail address—also called the top-level domain.

The bug wasn't an obvious problem until 2011, when ICANN decided to dramatically expand the range of what can appear to the right of the dot (see "[ICANN's Boondoggle](#)"). Between 2012 and 2016, the number of top-level domains ballooned from 12 to over 1,200. That includes 100 "internationalized" domains that feature a non-Latin script or Latin-alphabet characters with diacritics, like an umlaut (¨),

or ligatures, like the German Eszett (ß). Some 2.6 million internationalized domain names have been registered under the new top-level domains, largely concentrated in the Russian and Chinese languages, according to the new study.

Many Web applications or e-mail clients recognize top-level domains as valid only if they are composed of characters that can be encoded using American Standard Code for Information Interchange, or ASCII. The problem is most pronounced with e-mail addresses, which are required credentials for accessing online bank accounts and social media pages in addition to sending messages. In 2016, the group tested e-mail addresses with non-Latin characters to the right of the dot and found acceptance rates of less than 20 percent.

The bug fix, which entails changing the fundamental rules that validate domains so that they accept Unicode, a different standard for encoding text that works for many more languages, is relatively straightforward, says [Ram Mohan](#), the steering group's chair. The new research suggests that the potential economic benefits of making the fix outweigh the costs. Too many businesses, including e-commerce firms, e-mail services, and banks, simply aren't yet aware that their systems don't accept these new domains, says Mohan.

Things are improving, though. In 2014, Google updated Gmail to accept and display internationalized domain names without having to rely on an inconvenient workaround that translated the characters into ASCII. Microsoft is in the process of updating its e-mail systems, which include Outlook clients and its cloud-based service, to accept internationalized domain names and e-mail addresses.

It's not just about the bottom line, says [Mark Svancarek](#), a program manager for customer and partner experience at Microsoft, and a vice chair of the Universal Acceptance Steering Group. To let millions of people be held back from the Internet because "the character set is gibberish to them" is antithetical to his company's mission, he says.

Acceptance of non-ASCII domains is likely to spur Internet adoption, since a large portion of the next billion people projected to connect to the Internet predominantly speak and write only in their local languages, says Mohan. Providing accessibility to these people will depend in many ways on the basic assumptions governing the core functions of the Internet, he says. "The problem here is that in some ways this is lazy programming, and because it's lazy programming, it's easy to replace it with better programming."

Note: This news has been originally published in [MIT Technology Review](#).

UNSUNG HEROES

INTERVIEW

Zoltan is one of the passionate tester I met via our adventures with BBST Foundations class by AST.

Since then we have taught the class together on multiple occasions and have also been talking testing once in a while. This humble boy, originally from a small village of Hungary has over 20 years of testing experience and has been a lone warrior of CDT in Hungary.

I would describe Zoltan as a bit of perfectionist, as he usually sets very high standards for himself, especially in testing that he loves.

He likes CDT mostly because intellectually it is very-very rich and deep. He likes that it does not want to provide universal truths, unified definitions and that it emphasizes the interdisciplinary characteristics of testing.

On his trip to Germany, we met and talked about testing over coffee. The interview is sort of outcome of it... if you like you can reach out to him via Twitter @zzmolnar or Skype: zzmolnar.

Read and enjoy!

- Lalitkumar Bhamare



Over A Cup of Tea

With Zoltán Molnár

Thanks for doing interview with me Zoltan. How are you doing?

It's my honour to be interviewed by your magazine, Lalit. I am fine, loaded with plans and tons of work.

As far as I remember, we first met online for AST's BBST class. I'm curious to know your testing journey (and Context Driven Awakening in particular)

You seem to have a great memory.

I started to work as a tester in 1998. I accidentally saw a testers' job advertisement during my last semester and although I did not know anything about software testing, I did not particularly like programming either. I thought my fault tolerance related major would become a good reference to the position. I applied for the job and got hired.

I had been a factory school minded tester for more than a decade then, created and run a lot of scripts, wrote numerous test plan and test case description documents. I visited the first testing course of my life in 2001, acquired ISTQB certificates a few years later when the company jumped on that educational bandwagon. I think of myself as an inquisitive person who likes reading, I had subscribed for a few online tech magazines and occasionally browsed for testing related publications. So if I look back, I consider this period of my life as a reminder that I need to be a more efficient researcher.

I slowly become a disillusioned, slightly bored tester but in the beginning of 2011 I faced a Bach and Bolton interview in the Agile Record magazine. The interview felt extremely inspirational, I got immediately hooked. I felt like Neo once the world beyond the Matrix had been opened to him.

That article became the door to the context driven community. I quickly realized that even if I have advanced level testing certifications, I still have plenty to learn (and much stuff that I was taught would be better to forget). I plunged into online research as I hadn't recognized any Hungarian tester with CDT interest. I found the sometime software testing group on Yahoo, started to take the BBST courses of AST and participated on Weekend Testing events. I even took an online Rapid Testing Intensive class next year. I tried to follow discussions and debates on different forums, occasionally asked for skype coaching from some of the respected members of the context-driven community too.

My professional attitude has also changed in the meanwhile. I am an introvert but I started to question louder the widely accepted processes that I thought to be wasteful or harmful to the product development at the company that I was working for, tried different formats of lightweight test documentation on my own, changed the way I communicate about testing and report the results, urged tester fellows to stop investments into misleading measures and wasteful administrative systems. I frequently challenged the testing leads' ideas. I became a rebel. My first meetup presentation targeted traditional basic testing beliefs and the value of certifications that ended in an open space debate with an associate professor. Behind my back I was called 'Che Guevara' by some of my colleagues.

I also had to get over some thought barriers. I read a lot for example a lot about the inefficiency of scripted testing, but in the meanwhile I was working in a context where use of script languages was simply vital to perform even mediocre testing. It took some time till I figured out that meaning of the 'script' term is severely limited in my mind.

I cooled down a bit in the last years. My thinking become more matured but I never gave up learning – and started teaching. I have been a BBST instructor of AST for three years, of Altom since the last year. And one of my wishes came through too: I could make the RST courses in Cluj, Romania, paid on my own cost.

From how much I know you, you always try to make things better. Have you ever had any tough times while trying to change things for better? What are the lessons learned?

Yes and I think tougher periods are inevitable in our life. I have to admit I haven't been very active in the last couple of months.

I felt it is high time to reassess my abilities as a responsible tester and teacher due to some professional failures. I think your readers mostly read and hear success stories so please let me to go into the details a bit. Although I tried several times, I failed to sell either the BBST or RST courses to the company I work for. That makes me wonder if either my communication or marketing skills need improvement perhaps or in fact -in spite of their frequent “our testing is ineffective/expensive” rants- the good old factory school approach seems to satisfy both the engineers and management. Whatever it is, it is disappointing to me.

Another story, I failed two years ago as a project test lead. Once I was appointed, I felt it was high time to step out of the traditional, unified processes that test teams follow, especially because both the project and product context was vastly different to the ones we were used to. I set up a bare minimum set of rules for myself that led to confrontations then a break with my superior after a couple of months. I think I have overestimated the freedom and authority I had as well as my capability to drive the changes in a deeply hierarchic system. It was a good lesson and just motivates me to become even better.

At last, I became disillusioned with my very own, short, introductory testing training at the company that I had run for years. Obviously, no one can be taught to test in a day, but as I see now, I focused to wrong place. I should revise my approach and perhaps rework the training thoroughly.

How is testing community in Hungary doing? Any interesting things to share?

I don't feel it very lively. I am aware of a few meetups, the most attended one is organized by the Hungarian branch of ISTQB as well as the biggest testing conference once a year. I prefer to visit a less popular meetup to meet and talk with some self-learners, but I have to admit I slightly neglected them recently. Please let me to name two testers with very strong CDT interest here: both Lina Zubitye (@buggylina) and Erik Hörömpöli (@erikhun) are passionate, keen to learn and work hard on their skills to become a great tester. Did I say they are much better community builders than me? And they occasionally blog, please pay attention to them!

Any testing professionals that changed your life as tester? How?

Cem Kaner, Michael Bolton and James Bach. Anne-Marie Charrett has also coached me twice over skype. I am grateful to all of them for opening my mind and giving back the passion to my profession.

What are your ways to keep improving your testing skills?

Skills do not improve without practice and occasional failures, I reckon. No one becomes an excellent driver by learning the traffic rule book inside out, reading car racing blogs or watching streamed Formula-1 races.

I used to participate on a number of Weekend Testing sessions that helped to think out of my box for a few hours. I like debates to improve my communication and arguing skills.

I try to actively apply what learned, whether it is management by mindmaps, use of safety language or different kind of fault reporting practices. I draw my own conclusions from the experiences at the end, but I admit I really miss others' feedback or coaching sometimes. I also try to spend some extra time on studying the technology that is valuable at my workplace. Currently it is containerization over virtual platforms and object oriented python scripting.

What resources you rely on to get better at your testing skills? (Books, blogs, magazines, forums etc.)

I read a lot. I have bought more testing related books yet that I read so far. I would not call any of those the changer of my life but I have some personal favorite of course. I loved Perfect Software and Are Your Lights On? from Weinberg. I am absolutely convinced that BBST workbooks are the best educational material one can get. Right now, you would find Kahneman's Fast and Slow Thinking on my night stand. I also have a nice collection of testing related articles that I considered worthwhile to keep downloaded on my disk. It counts more than three hundred, mostly CDT related files up to now and I look for those first if I dig myself into a specific topic occasionally.

What comes to social activities, I used to follow dozens of blogs, now it is reduced to a few only. I am a passive twitter user, rather just run through the discussions once per day.

Please tell me your top three lessons learned as a Student of Testing

Ranking the lessons would be hard, I would just say three that definitely left mark:

- All oracles are heuristic. They provide partial and sometimes misleading information. As I see, surprisingly many practitioners do not understand its implications.
- Automation is nothing more but using tools. The manual/automated testing distinction is terribly misleading.
- Scripts are results of exploratory activities. One of the striking revelations from Michael Bolton's RST course.

What is your feeling/opinion about current state of testing?

As I see the industry slowly tries to adapt itself to the latest technological improvements. For example, cloud computing and virtualization were basically unknown terms a decade ago, now these help to solve the price, availability or accessibility problem of SUTs for a lot of companies or allow the save and reuse of system states either for pre-configuration or troubleshooting purposes for the engineers. The new application of new tools and practices usually trigger procedural and organizational changes too and testers start to look for their place in this changing world. Some people consider the tester role to be dead but it sounds weird to me. Just because we realize that more of our new tools requires programming skillset or that testers are not worth to separate from development and operating personnel, there is still need for experimenting the product.

Furthermore, I think standards, maturity and quality models still acquire more attention and significance than needed. I looked deeply into ISO 29119 and TL9000 (that is a telecommunication specific ISO 9000 derivative) and I saw how these can limit testers' effectiveness in practice. I can see hardly any (if any) professional arguments beside the existence of testing standards. Testing education also disappoints me (at least in Hungary, I have not much information about the rest of the world). What I perceive is just simple adaptation of some ISTQB syllabuses without any additional research or critics as if it would be someone true paradigm to follow. I form this opinion based on checking some local educational outlines and participating on dozens of hiring interviews that I made with graduates in the last few years.

You have been doing great things for community by teaching testing. What more can we expect from you this year?

I usually teach two BBST courses per year. I occasionally pop up on WT events and also plan to be much more active on the local meetups.

What can we find Zoltan doing when he is not testing?

Most probably either practicing Transylvanian folk dances, working in the garden or reading contemporary novels.

What message would you like to leave for our tester friends?

Take the responsibility for your own education and preserve your integrity. As one of my favourite quotes says, "Integrity without knowledge is weak and useless, knowledge without integrity is dangerous and dreadful."

Tea & Testing



with

Jerry Weinberg

What Is Quality? Why Is It Important?

Another Story About Quality

To test our understanding of this definition, as well as its applicability, let's read another story:

One of the favorite pastimes of my youth was playing cribbage with my father. Cribbage is a card game, invented by the poet Sir John Suckling, very popular in some regions of the world, but essentially unknown in others. After my father died, I missed playing cribbage with him and was hard pressed to find a regular partner. Consequently, I was delighted to discover a shareware cribbage program for the Macintosh, "Precision Cribbage" by Doug Brent, of San Jose, CA.

Precision Cribbage was a rather nicely engineered piece of software, I thought, especially when compared with the great majority of shareware. I was especially pleased to find that it gave me a good game, but wasn't good enough to beat me more than 1 or 2 games out of 10. Therefore, I sent Doug the requested postcard from my home town as a shareware fee and played many happy games.

After a while, though, I discovered two clear errors in the scoring algorithm of Precision Cribbage. One was an intermittent failure to count correctly hands with three cards of one denomination and two of another (a "full house," in poker terminology).

This was clearly an unintentional flaw, because sometimes such hands were counted correctly.

The second error, however, may have been a misunderstanding of the scoring rules (which were certainly part of the "requirements" for a program that purported to play a card game). It had to do with counting hands that had three cards of the same suit when a fourth card of that suit was cut. In this case, I could actually prove mathematically that the algorithm was incorrect.

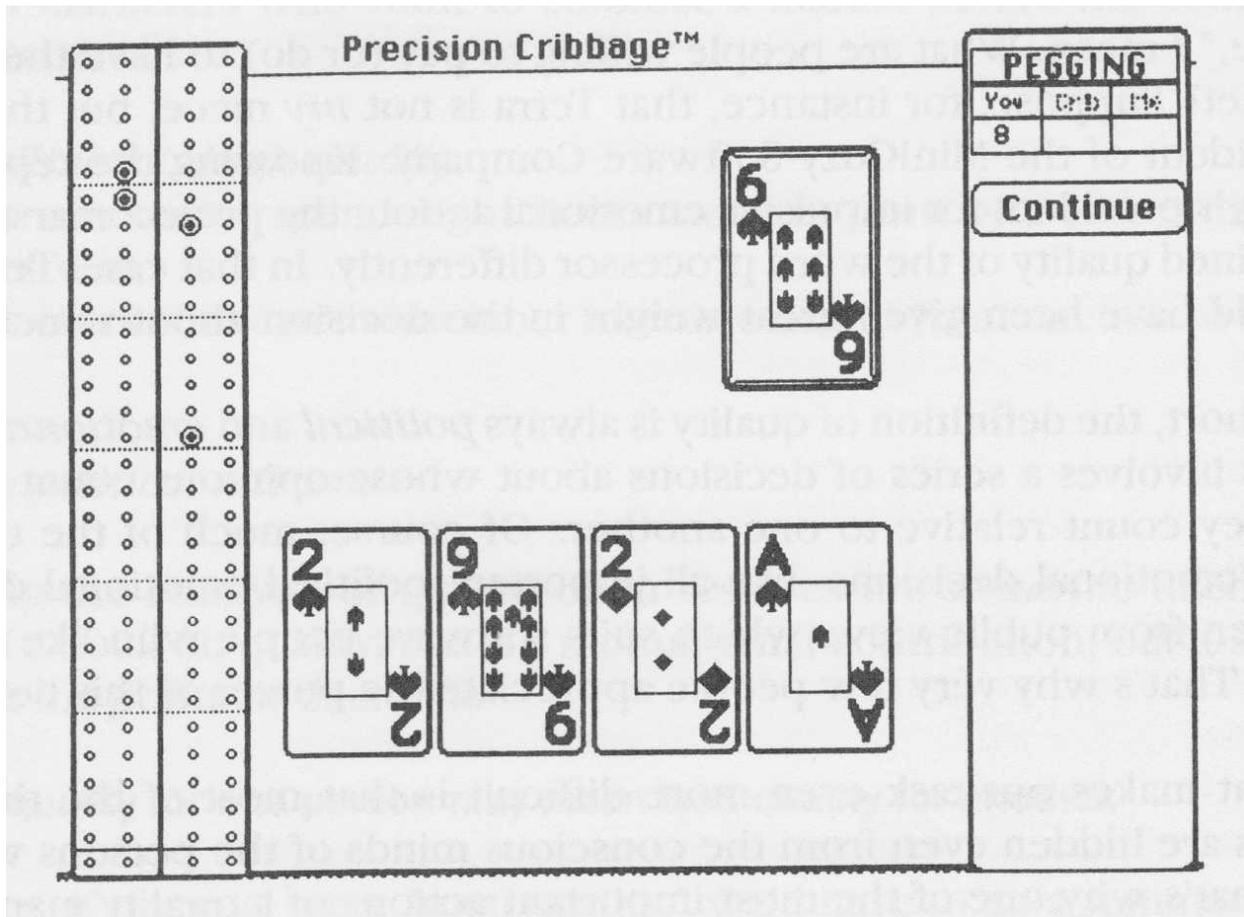


Figure 1-2. An example of a miscounted cribbage hand. The correct score should be 4, not 8.

What makes this story relevant is that even with two scoring errors in the game, I was sufficiently satisfied with the quality of Precision Cribbage to

- keep on playing it, for at least several of my valuable hours each week
- pay the shareware "fee," even though I could have omitted payment with no fear of retribution of any kind

In short, Precision Cribbage had great value to me, value which I was willing and able to demonstrate by spending my own time and (if requested) money. Moreover, Doug's correction of these errors would have added very little to the value of the software.

Why Improving Quality Is So Difficult

The tale of Precision Cribbage demonstrates that "meeting requirements" is not an adequate definition of quality, unless you're willing to accept a most unconventional definition of "requirements." It also demonstrates the inadequacy of definitions of quality based on errors, such as, "Quality is the absence of error."

Such definitions are easy to refute, yet they have dominated thinking about quality software for many years. This makes it easy for software developers and managers to ignore requests to "improve software quality." But don't they want to improve quality, even if nobody else was asking for it, just to satisfy their own pride? Of course they do, but nothing happens. Why not?

"It's not too bad."

The stories of CosyWrite and Precision Cribbage are typical of hundreds of cases I could cite, and you could undoubtedly supply many examples of your own. If you asked the developers, "Are you interested in a high-quality product?" I'm sure their professional pride would supply the answer, "Of course!"

But suppose you asked specifically about improving CosyWrite or Precision Cribbage. They would reply, "But it already is a good quality product. Of course it has bugs, but all software has bugs. Besides, it's better than the competition." And, of course, all three of these statements are provably correct:

- People are using their product, and are happy with it, so it is "good quality."
- All software does have bugs. At least we can't prove otherwise.
- People buy it over the competition, so it must be better, in their opinion.

Under the circumstances, there's very little motivation to improve the quality unless pushed from outside. If people stop using their product, or buying it, then the developers might decide to "improve quality," but by then it will probably be too late. Organizations that sell software simply fade away when faced with a competitor that operates in a more effective manner.

Organizations that produce software internally for larger organizations have little competition, so they simply stagnate. Whether or not their stagnation matters depends on what their parent organization defines as "quality." If the parent gets the value they need, and don't know any better, then the stagnation continues. Once they become dissatisfied, however, a crisis begins.

"It's not possible."

Did you know that if you were 8'6" tall you could get a job as a starting center in the NBA and earn \$5,000,000 a year? Now that you know that, why aren't you starting on a growth program? It's a silly question, because you don't know how to grow several feet taller.

Did you know that by reducing the faults in your software to less than 1 in a million lines of code you could increase your market by \$5,000,000 a year? Now that you know that, why aren't you starting on a quality program? It's a silly question, because you don't know how to reduce software faults to less than 1 in a million lines of code.

Phil Crosby, in *Quality is Free*, says that the motivation for improving quality always starts with a study of the "cost of quality." (I would prefer the term, "value of quality," but it's the same idea.) In my consulting, I frequently talk to managers who seem obsessed with cutting the cost of software, or

reducing development time, but I seldom find a manager obsessed with improving quality. It's easy for them to tell me what it's worth to cut costs or expedite a schedule, but the value of improved quality seems to be something they've never thought of measuring.

Yet when I suggest measuring the value of quality, they often respond as if I told them to measure the value of growing to 8' 6". Why bother measuring the value of something that you don't have the slightest idea how to achieve? And why try to achieve something whose value you don't appreciate? Figure 1-3 shows this vicious cycle in the form of a "diagram of effects," a form of diagram we will explain later and use throughout this volume. For now, let's just concentrate on what it says about why improving quality is so difficult.

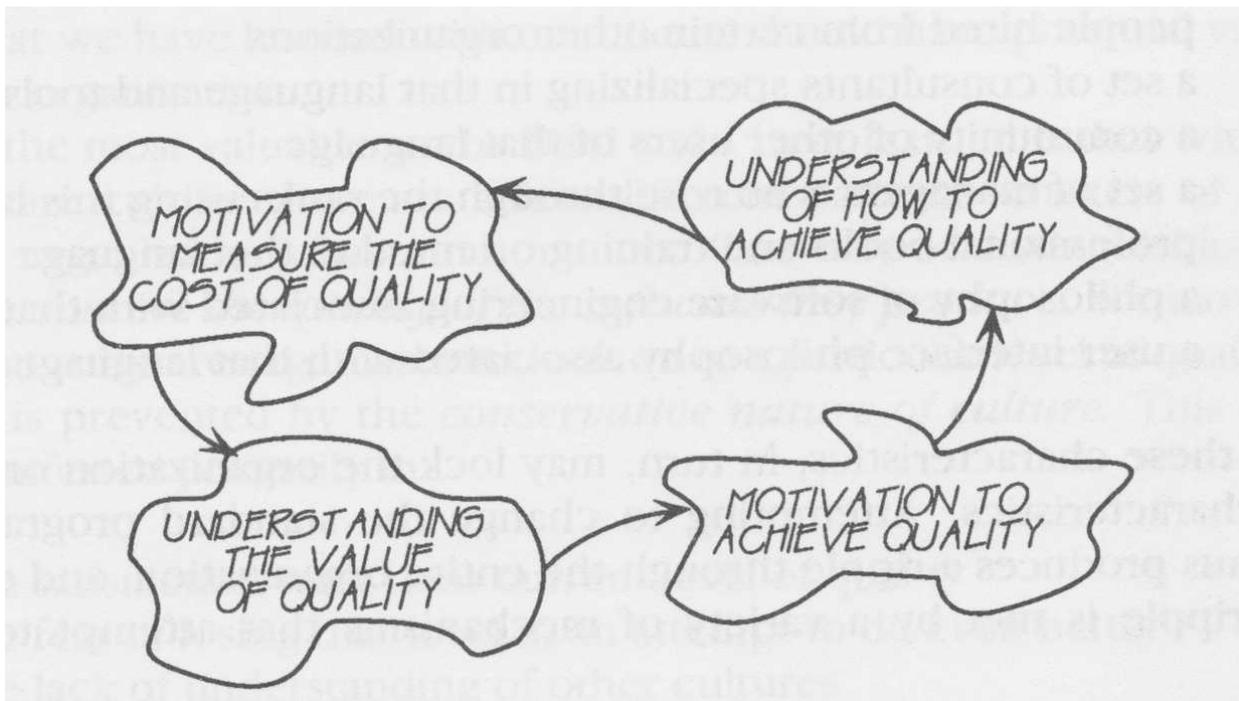


Figure 1-3. A vicious cycle that prevents organizations from starting to improve quality.

The diagram of Figure 1-3 can be read optimistically or pessimistically.

Optimistically, it says that once an organization begins to understand the true value of quality, its motivation to improve will rise, which will drive its understanding of how to improve, which will in turn lead to a better understanding of the value of quality. That's why Crosby likes to start organizational change with a "cost of quality" study.

Pessimistically, though, the cycle can be seen as inhibiting a change to higher quality. If there is no understanding of the value of quality, then there is no motivation to achieve quality, and thus no improvement in the understanding of how to achieve quality. And without knowing how to achieve quality, why would anyone try to measure its value?

The lock-on

Figure 1-3 happens to be a simple example of a "lock-on" effect. A locked-on system tends to hold itself to an existing pattern, even against "logical" reasons to change.

An excellent example of a lock-on is the choice of a standard programming language. Once an organization is using a single programming language—for whatever historical Reasons —the cost of changing increases, the motivation to study the value of other alternatives decreases, and the knowledge of how to obtain those alternatives disappears.

As a result, the organization "locks on" to the language, just as a country locks on to the side of the road used for driving.

In this volume, we'll see many examples of lock-on situations, but for now we simply want to note that lock-ons occur in clusters. When you lock on to a particular programming language, you also tend to lock on to some or all of the following:

1. a set of software tools supporting that language
2. hardware systems that support a particular dialect of that language
3. people trained in particular schools
4. people hired from certain other organizations
5. a set of consultants specializing in that language and tools
6. a community of other users of that language
7. a set of managers who rose through the ranks using this language
8. professional books and training oriented to that language
9. a philosophy of software engineering associated with that language
10. a user interface philosophy associated with that language

Each of these characteristics, in turn, may lock the organization onto another set of characteristics. Attempting to change the standard programming language thus produces a ripple through the entire organization, and each furrow of this ripple is met by a variety of mechanisms that attempt to prevent a change.

It doesn't matter that changing the language would be "good" for the organization.

As Virginia Satir, the family therapist, used to say "People will always choose the familiar over the comfortable."

Software Culture and Subcultures

All these interrelated lock-ons produce patterns. Every time Dani (my anthropologist partner) and I arrive at a new organization to consult on managing software organizations, we quickly notice two essential facts:

1. No two software organizations are exactly alike.
2. No two software organizations are entirely different.

Because of (1), it's not possible to have off-the-rack solutions to really important problems of software management, but because of (2), we don't have to start over from scratch with every new organization. There are commonalities from one software organization to another, even though they are different sizes,

in different industries, working with different programming languages, in different countries, and even in different decades. This book is very much concerned with those commonalities.

A rather anthropological way of expressing this observation is that there is some sort of "software culture" that transcends boundaries of time, space, and circumstance. There are a number of ways to verify the existence of this "software culture." For one thing, software books sell very well in English all over the world. The software culture is very much an English-language culture. The books also sell well in translation, and the same software jokes—like Levine, the genius tailor—are funny all over the world for decades.

Software meetings are international, and surveys of attendees show that they cross industries and age groups as well. We are fortunate that such a software culture exists, for it allows us to learn from one another. Thus, any truths that we have learned with our clients should have potential value for you and your organization.

One of the things most valuable truths we have learned is that the overall software culture seems to come in a few different patterns—clusters of characteristics they have in common. One way of distinguishing these patterns is to observe the quality of the software they produce. We have come to believe that software organizations lock on to a particular level of quality, and that change is prevented by the "conservative nature of culture." This conservatism is manifested primarily in

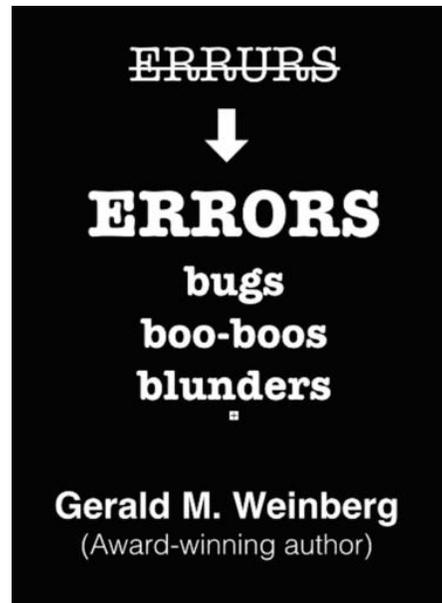
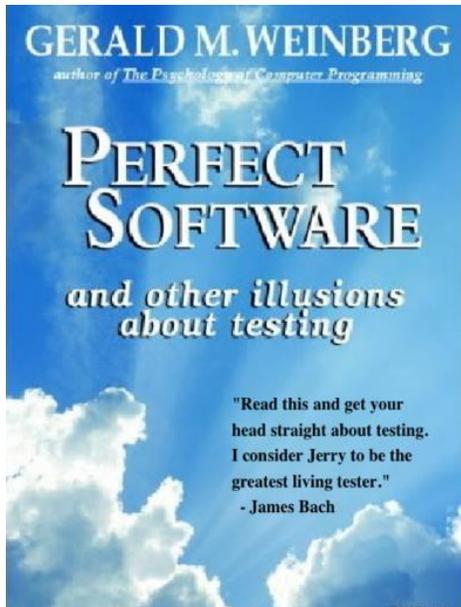
- the satisfaction with their current level of quality
- the fear of losing that level in an attempt to do even better
- the lack of understanding of other cultures
- the invisibility of their own culture

Quality is important because quality is value. The ability to control quality is the ability to control the value of your software efforts. To reach a new culture of quality software, developers and managers must learn to deal effectively with these factors.

To be continued...



If you want to learn more about testing and quality, take a look at some of Jerry's books, such as:



People Skills—Soft but Difficult



Curious to know why you should get 'People Skills' bundle by Jerry? [Then check this out](#)

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 **SoftwareTest 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 Weinberg has been observing software development for more than 50 years.

Lately, he's been observing the Agile movement, and he's offering an evolving set of impressions of where it came from, where it is now, and where it's going. If you are doing things Agile way or are curious about it, **this book** is for you.

Know more about Jerry's writing on software on **his website**.

AGILE IMPRESSIONS



Gerald M. Weinberg

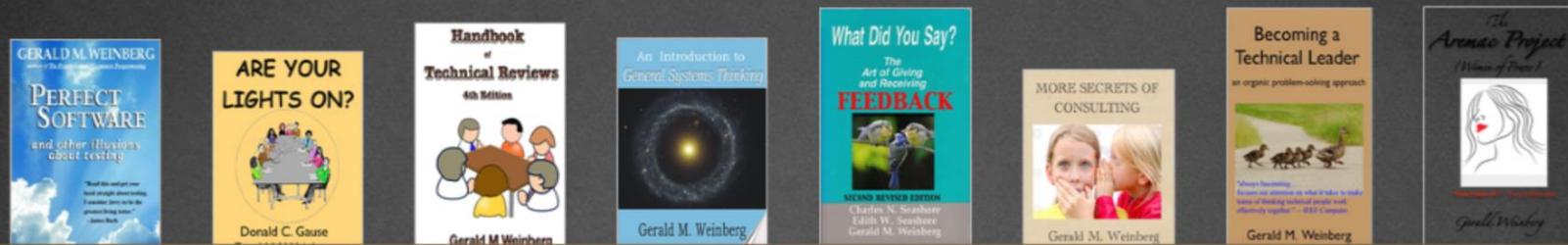
TTWT Rating: ★★★★★

The Bundle of Bliss

Buy Jerry Weinberg's all testing related books in one bundle and at unbelievable price!

The Tester's Library

Sold separately, these books have a minimum price of \$83.92 and a suggested price of \$83.92...



The suggested bundle price is \$49.99, and the minimum bundle price is...

\$49.99!

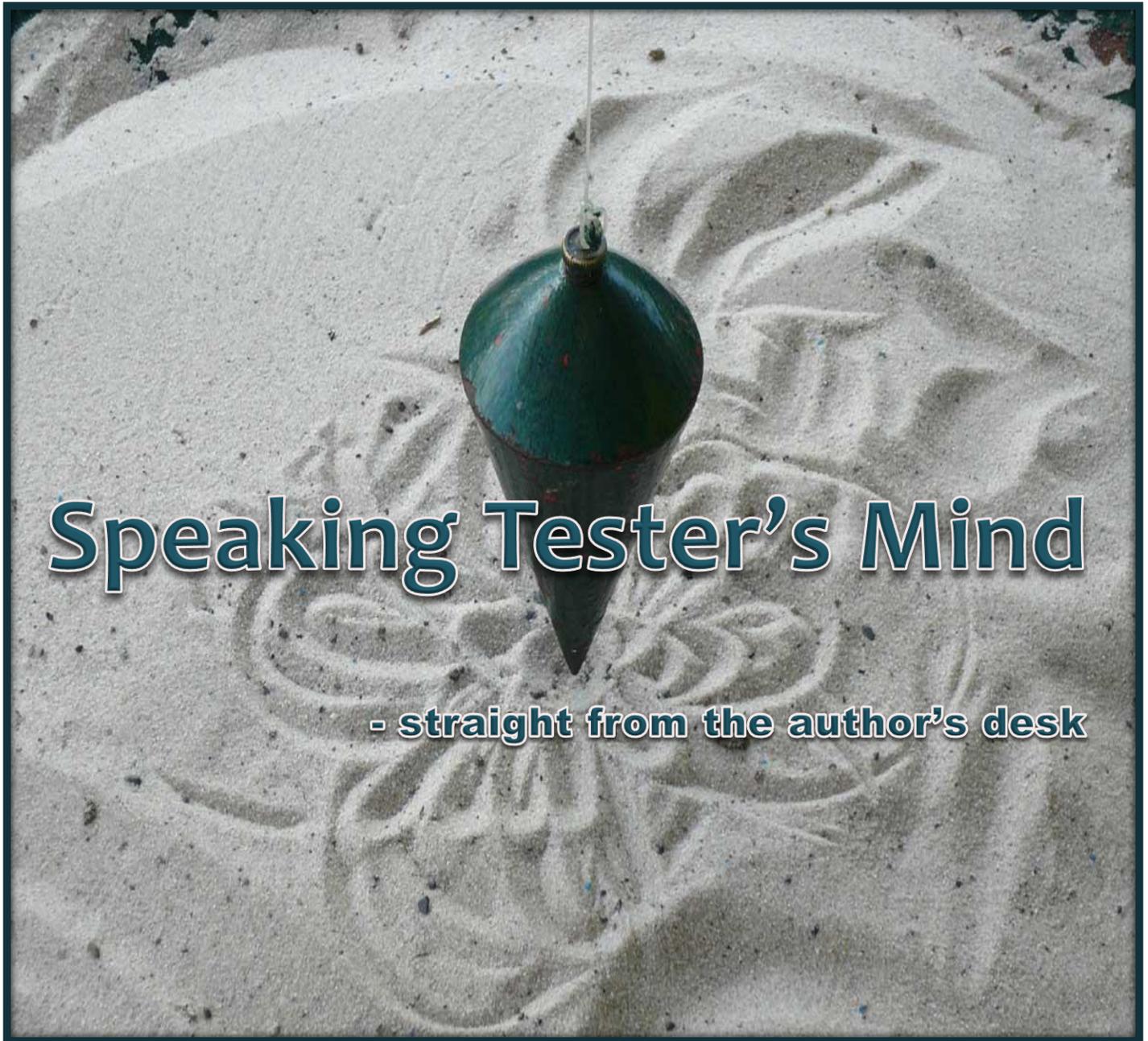
Buy the bundle now!

The Tester's Library consists of eight five-star books that every software tester should read and re-read. As bound books, this collection would cost over \$200. Even as e-books, their price would exceed \$80, but in this bundle, their cost is only \$49.99.

The 8 books are as follows:

- Perfect Software
- Are Your Lights On?
- Handbook of Technical Reviews (4th ed.)
- An Introduction to General Systems Thinking
- What Did You Say? The Art of Giving and Receiving Feedback
- More Secrets of Consulting
- Becoming a Technical Leader
- The Aremac Project

Know more about this bundle

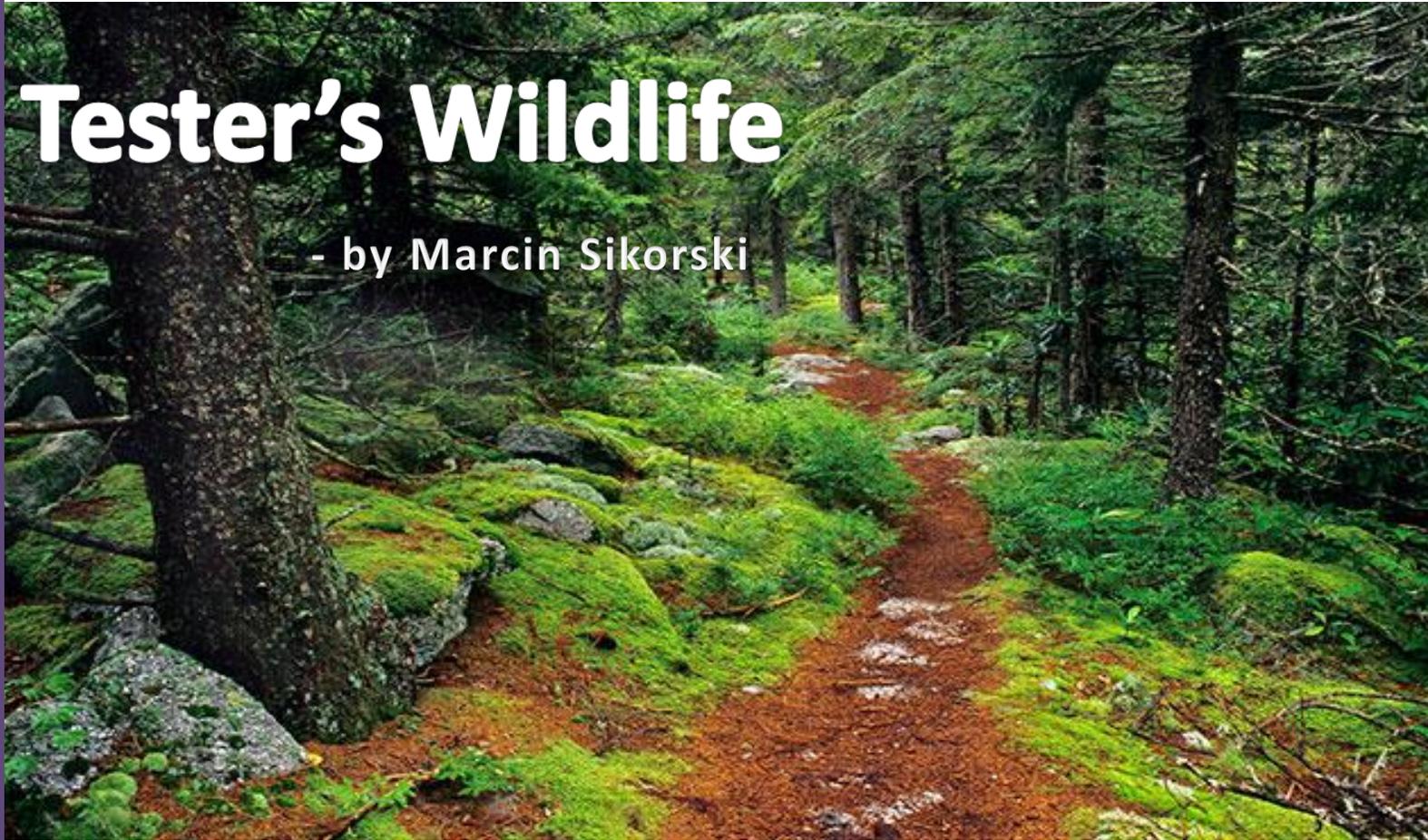


Speaking Tester's Mind

- straight from the author's desk

Tester's Wildlife

- by Marcin Sikorski



Putting your feet into the wild, corporation jungle might be intimidating at first. Environment is strange, huge and full of mysterious signs, colors and elements. You hear plenty of loud noises coming from all over the place – making it difficult to focus. Cables are laying on floor like wild snakes just waiting to be stepped on and there is a huge chance that at some point you will encounter crowded pasture full of coffee machines. Such dappled sight and piercing noise attacks your senses making you wonder how anyone could survive in such peculiar environment, not to mention – why? And then, all of a sudden, you meet creatures known as IT beings and everything starts to make sense.

You see I'm one of those animals - known to outside world as a Tester - who is struggling each day to coexist peacefully with a separate, wild specie called Developers. Our lives and encounters are usually loud, bloody and full of unexpected plot twists but, you see, they are unfortunately necessary for the product to be delivered to the Client. Our day to day battles can be stressful and difficult at the same time but you wouldn't expect nothing less dramatic from clash between lions and elephants, would you?

And this is where our story begins.

Due to the fact that some people tend to believe life in IT Corporation is only full of rainbow and sunshine let's pretend for a minute we are David Attenborough and spend few next minutes on discovering – how one day of wild Tester looks like?

Testers wake up irregularly and as result they also show up at different period of day. Some get out of the burrow at 7 am and some appear at 10 – it depends on day of the year and specific male. All of them, without any exception, will however start there day by meeting near the watering place full of coffee. They will interact with each other by using verbal noises full of laughs. It seems this is the time they fell most comfortable and in fact it is the only time of a day when they will fully charge their batteries.

After this initial meeting those creatures will eventually come back to their nests full of prototypes, test devices and test documentations. Each nest may be located in different part of the office as this depends strictly on territory of the project they are assigned to guard. Each territory is represented however by obscure forest called open space. Place where time slows down and energy of young brutes is transformed into lines of code. The biggest Testers' blocker comes however in form of watch duty full of unexpected twists due to the fact the same territory is occupied by wild Developers. And believe me – tension between those two is solid especially when there is one Tester versus dozens of them.

Just spending few minutes on silent observation leads to first encounter initialized by one of the code makers. You see at some point, without any signal, one of them will stand up, move silently from his coding land and come close near Tester with one intention – laying his paws on small and shiny thing known as DUT (Device under test). Of course he won't bother saying why he wants that item but he will repeat like mantra one sentence "let me debug the problem". This behavior is known as JIRA reproduction. You see each of the alfa males is assigned to one, specific task. This means they need to take care of this job unless they want to be kicked out of the herd. As they don't trust Testers - who they recognize as their arch nemesis – they command to give them the DUT so they can perform war dance called "local problem debugging". This situation leads however to voice conflict and act of domination which is usually solved by strange, mysterious creature called PM which all animals seems to be afraid in the same way. Anyway DUT is transferred from one side to another and the mentioned Developer will celebrate this by doing small hand shake indicating "It will only take 20 minutes to fix the problem".

It goes without saying that it never does but I've discovered fascinating fact – all of IT creatures are unable to specify exact periods of time when it comes to saying how much time something will take. Even if they mark their territory by booking meetings or deadlines. What is even more striking is the fact that Developers have strange tendency to never come back once they borrowed the property and hide in the nest. But hey, at least they do keep the word and provide patch that fixes the problem. And additionally adds regression to five new features. But have no worries - this will be fixed ASAP once new ticket is created. Thus, new Tester will be hunted and the story will begin all over again.

Speaking of which if we take closer look at Testers as a group we can notice that most of them sticks together, covering each other back and making sure they focus all their energy to achieve PASS results. This leads to cooperation, splitting tasks and overall doing their best. However, not all of them seems to share the same passion as once in a blue moon there comes this one black sheep that really can't help to grasp the idea of team work and tries to be really annoying. This sheep – which usually, for some unknown reason, is an extra contractor working on the other side of the globe - seems to always see some kind of trouble even if it doesn't exist. He alarms all members of the herd by reopening once closed tickets (even if it means using obsolete, budget, 3rd party devices) and usually panics by saying "Problem is reproduced 10/10 times" even if no one else had similar issue for a long (if ever) time. Conversation between two sides of the testing teams, living on two separate parts of the world resembles ping pong match where JIRA comments create mountain of text so huge that at the end nobody even remembers what the case was. What can we say? Animal kingdom is puzzling place very difficult to explain.

Rest of the day usually goes smoothly for the quality makers – they hunt for sweet donuts on coffee corner meetings, fetch some test summaries, stroll to lab to do some specialist checking and, what is most frustrating for them, they try to prepare setup to reproduce specific bug coming from creatures called users. And boy, oh boy, this is really magnificent moment to behold. Flashing the phones, updating the firmware, adding the binary files – it is tedious yet doable part but making sure that something works with environment like Linux/Windows/Mac/Android is all new level of abstraction. Downloading latest updates, sudoing packages, downgrading the versions, reverting changes or adding that funny little line in system file using VIM just to discover something is no longer supported or backward compatible takes most of testers' energy and passion. It also costs sweat, pain and time but trust me – even if reflecting

user's land might sound like unproductive, meaningless thing to do this is a must. This is how quality and flawless products are being born and this is why nature is not always pretty.

Yet with all the wildness and unpredictability of land where Managers and other predators are waiting for your mistake by asking difficult questions like "Can you estimate this problem for the deadline?" and place where simple camouflage of "Strange, it worked for me" decides between saving your life or being sacrificed on next SCRUM's meeting it is still the greatest place to live. Community, besides what I've just said, is usually nice, completing tasks bring wild – no pun intended - pleasure and each new day is more interesting and beautiful than previous one. Granted, every aspect of life of humble IT creatures is fascinating at the same level and few disadvantages can't overshadow the purity of their work.

Question remains – will you be brave enough to become part of this jungle and if so how long will you survive?



Marcin Sikorski (email: marcin.sikorski@test-engineer.pl | twitter: [@TestingPole](https://twitter.com/TestingPole)) is currently working as Test Engineer for wearable division at Tieto.

He is a mobile testing enthusiast actively involved in Smart devices industry

Teach Them All to Code



Despite the widely held belief that programming knowledge is unnecessary in software testing, I argue in this article that **all** testers should be able to write code. Testers with no programming skills are far less qualified than the ones that possess them. Testers' ability to write code is such a fundamental aspect of their everyday work that the inability to program is akin to modern illiteracy. In addition, I posit that writing code is just one skill among many that testers should apply while investigating a product under testing; a competent tester should carry out "manual testing" as well as "automated testing" at both the business and technical levels. They are all part of the same activity: exploring the product and seeking to gain understanding of its quality features. Finally, I argue that learning how to code is not as difficult a task as it may seem to some. Not all testers need to graduate from SW engineering or computer science programs. In fact, writing code using a high level language (such as Java, C#, etc.) to achieve a predefined goal while testing a product is fairly simple to master.

The Challenge

In the 2016 issue of the [State of Testing](#) Report, over 1000 participants worldwide were asked how much they value the importance of a set of professional skills presented to them. One of the questions asked was "What skills do you need to be a good tester?" Only 25% indicated that programming skills were very important. 56% stated that it is an important skill, and 19% stated it is not important at all. This means that almost 50% of testers do not consider programming skills to be essential skill in their work! Furthermore, in the report, programming skills were considered equally important as business skills, **less important** than customer-facing skills and **far less important** than communication skills. These are people's opinions. So, the question still stands: How important are they really?

Can we test software without understanding what it is made of?

The definition of software testing (based on [Cem Kaner's definition](#) from 2006) is "an investigation conducted to provide stakeholders with information about quality of the product or service under test." Being able to provide valuable information consists of many layers. Two of them can be outlined

immediately: the business layer, which considers the system level requirements and deals with the question “does the system do what it was designed to do?”; and the technical layer, which deals with verifying that a piece of code works correctly and with minimal side effects or performance issues. In order to gain thorough knowledge of the test subject, testers must do at least the following:

- Review the test basis related to the system under test - this would be any kind of requirement material - both technical and non-technical.
- Inquire about details from the people who planned and built the product.
- Perform checks on the system under test to evaluate its behavior under various conditions.

Can an investigation be complete without some understanding of the building blocks of the product? Imagine, for example, an organization hired to evaluate the quality of a car that has minimal knowledge about how a car is built. Or a civil engineer who does not know the basics of constructing a building. The basic understanding in the work that created the product we are evaluating is essential to evaluate it thoroughly. The quality of the investigation process, from test case design to their execution lies mostly on the testers’ common sense and ability to challenge the system in question and its creators. Does it mean we have to be familiar with the production code of the test subject? Or be familiar with every technology being used to build the product? I don’t believe so. Not only it is impractical, but unnecessary. Being familiar with at least one programming language (preferably an Object-Oriented one) and understanding the foundation of computer programming simply provides us with a set of tools to gain better technical understanding of the creation process of the product we evaluate.

At this point I assume some of you may think this notion is trivial. If this is the case, then why are there so many testers who believe that code writing skills are not of high importance? Or that there are more testers out there who can’t program than ones that can?

I believe many testers do not have programming skills for 3 main reasons:

1. Testing is an evolving profession, without a well formed standard like computer science or SW engineering. Therefore it is not clear what the skills set of a tester should be.
2. Either testers or their employers do not consider testing activities as such that require deep technical skills, so they focus on the business aspects of the product, not realizing the major effect the technical aspects has on the quality of the final product.
3. Testing which involves code is considered a job for skilled employees, such as SW engineers.

These 3 reasons, along with rapid changes in technology have created two types of testers - the “manual” tester and the “automated” tester, where automated testing is considered a task for higher skilled employees. Quite a few are professional programmers. For the less skilled testers, great effort is still being made to create testing tools which do not require any programming skills. Let us examine an example:

The Case of Record and Replay - Microsoft Coded UI Framework

Till this day, companies have invested a lot of time and effort into building tools to perform error checking without the need to write a single line of code. The [Microsoft Coded UI test automation tool](#) was originally introduced as part of visual studio in 2010, and its popularity has been growing ever since. One of its main and powerful features is to “... permit testers to quickly and effortlessly create coded UI tests by recording the actions performed...” Alongside Microsoft there are many others - HP UFT (previously QTP), TestPlant’s EggPlant test automation tools and many others.

The disadvantages of record and replay testing tools, such as maintenance issues and lack of flexibility, are well known and deeply discussed in many forums. You can even find a [basic discussion about it in the ISTQB syllabus](#). Even though it is claimed that these type of tests saves a lot of time (mainly by the companies who market these tools), at the end of the day record and replay test scenarios result in flaky tests. So why do we insist on working so hard to create test automation tools to address this lack of skill, instead of training testers in the fairly basic skill of programming? How hard could it be?

The Task of Training

For the past 8 months, I have been implementing “automated tests” in a company that employs testers without any coding abilities. One of the first tasks I had was to develop what is known as a “test framework”. Sitting down to build the framework, it was clear to me that the framework should be as simple as possible, so anyone with basic programming skills would be able to contribute to the testing of the system. In addition to creating a simple framework, I faced the task of teaching “manual testers” with no previous background of programming, basic knowledge, in order for them to use the infrastructure and build code-based automation scenarios. No KDT, no record & replay - just writing test scenarios using C#. We had a five-day crash course followed by given tasks of writing scenarios which were previously done manually. The results were surprising to all. Not only had the “manual testers” successfully written code based test scenarios after just 5 days of training, the quality of the scenarios far exceeded the “automated testers” since the business level knowledge that the “manual testers” had on the product exceeded their fellow “automated testers”.

Furthermore, since the task of writing an automated test scenario became easy and accessible, tests done by means of “exploratory testing” or “bug hunting” are immediately translated to a future automated test. Thus, an ongoing improvement was made the automated test suite without the need for mediators such as “automated testers” and the “process” of creating an automation script. The job of automation testing became easy.

Coding Skills Used During Exploratory Testing and Bug Hunting

But how does programming knowledge help us during manual testing? Since contemporary methodologies such as exploratory testing or bug hunting are extremely popular, hunting for bugs when armed with programming knowledge helps to reveal issues such as:

- Null pointer references
- Buffer or Array Overflow
- Other types of exceptions
- Memory leaks
- Resource related issues such as file handling, resource misuse because of concurrent racing for resources, etc.

Do you need coding skills in order to find these types of bugs? Not necessarily. Skilled testers may find them anyway; however, if you know how to look, it’s more likely these defects will be revealed, while others will simply stumble upon them by chance. Testing methodologies have tried to bridge these gaps using techniques such as boundary tests and others. But in my opinion, being a good tester is similar to being a good mechanic - you need to be able to tell something is wrong just by listening to the engine.

Coding vs. Modern illiteracy

Historically, the ability to read and write was not as commonly held as it is today. In fact, many people, including women and African-American slaves were forbidden from becoming literate. However absurd it may sound today, there are plenty of example of both historical and contemporary oppression of access to literacy. [African-American slaves in the United States](#) were not allowed to learn how to read and write by law since it "... [had] a tendency to excite dissatisfaction in their minds, and to produce insurrection and rebellion, to the manifest injury of the citizens of this State..." Women were barred from learning to read and write since it was considered a waste of their time; that they should be performing other tasks rather than reading. As recently as 2009 there was the case of [Malala Yousafzai](#), a young Pakistani girl who famously defied the Taliban as they attempted to prevent her from getting an education because she is female.

Since I liken lack of coding knowledge to modern illiteracy, I've been asking managers and fellow testers those exact same questions over the past few years: why not teach all testers how to code? Here are some of the answers I got:

1. Since they are not qualified for it, the time it will take them to perform the task well and thoroughly will simply not be efficient.
2. Automation tasks are doomed to fail, as it has for the most part of the last 20 years.
3. We don't need them to know how to code - we don't consider it an important skill
4. Since the salary of manual testers are not as high as automated testers, having a skilled team will unnecessarily increase the testing expenses...
5. They won't want to learn since it is a harder task than what they signed up for.
6. Since coding is a time consuming task, focusing on them will make manual testers overlook the basic business level errors that exist in the system.

Can you think of someone today that believes learning how to read and write is a waste of time?

Just like In the old days where Christian clerics insisted on reading the bible solely in Latin, these days there is a growing movement of companies who believe that testing should be done solely by the developers themselves, as part of their overall responsibility, and not employ any testers at all. While developers should be accountable for their code (as they say in Google - you build it, you break it) and should perform copious micro-testing to ensure they do not introduce new defects into the product, I believe the main reason for this approach is that these types of companies don't see the value of testing as a profession. I believe this is due to the fact that till this day there is still not a clear skill set of what is required to be a skilled tester, and there are too many 'manual testers' out there who just can't deliver.

Well, luckily for us the times are changing in this area as well. In times where elementary school students are already being taught basic programming using games such as [CodeMonkey](#), and middle schoolers are taught the art of becoming code-writers using [Arduino](#), I think it is safe to say that in the not so distant future, almost everyone will know the basics of computer programming in some form or another.

Conclusion and Final Note

As test leads, we need to push programming skills into our existing staff training. We need to build tools that will allow all of them to express themselves using code. We need to drop the notion that some testers are more skilled than others, when the difference truly lies in the ability to write code and not on solely on work performance.

We need to change the way we consider test automation tools as just another activity in our testing quest. If you design a test suite for a product, then using code to automate some of the tests should be a tool in your arsenal just like any other tool such as performance monitors, diff tools, etc. It can be only achieved by ongoing training for testers, pushing them high and beyond their current skills set.

We need more tech-oriented testers. We need more testers who can challenge programmers by speaking in their language - source code, rather than heavily rely on the business level requirements as the base of communication. This type of communication requires at least a basic understanding of computer programming. Tech-oriented testers will produce far better results than pure business-oriented testers.

In addition, programming is fun and challenging. I believe that testers who use code in their everyday work will feel challenged and satisfied by the fact they are evolving and becoming more professional. Their work product is improved, and the result is an overall increase in the quality of the system under testing. Don't we all deserve that?



Nir Gallner is the founder and CEO of VeriSoft Testing Services- a boutique software testing company whose mission is to help its customers implement a reliable test strategy that reduces product risks.

Nir started his testing career in 2002, and during that time, participated and lead numerous testing projects in various industries including defense, telecommunication and finance.

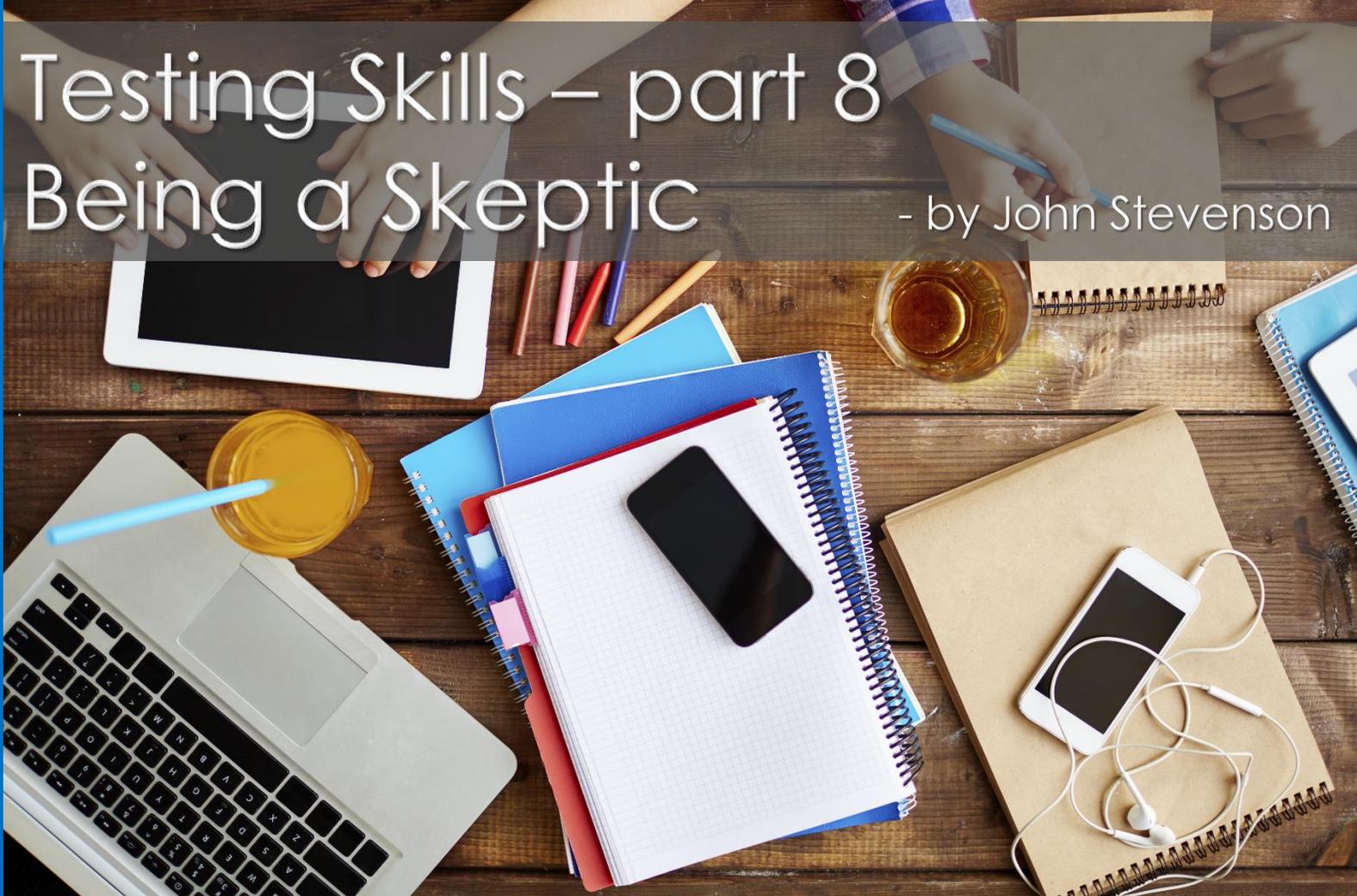
He describes his work as "breaking software and systems for a living" and has been doing that with great passion for over 15 years.

In addition, Nir is a lecturer in the field of software testing and sees it as his mission to shape the next generation of software testers. Contact Nir via email on nir@verisoft.co

Testing Skills – part 8

Being a Skeptic

- by John Stevenson



"Science is the organized skepticism in the reliability of expert opinion." Richard Feynman

The reason for this article is based upon a recent twitter conversation regarding how someone sometimes avoids skepticism due to its negative nature. I fully understand what they mean when skepticism is used in way to attack a person or their character ([ad-hominem argument](#)). The purpose of this article is to provide information about how useful having a healthy dose of skepticism is for those involved in software testing.

Many people have a belief that the purpose of testing is to prove that the software works as expected. This is really a fallacy since testing cannot prove that the software will **always** work the way you expect. There is always an element of doubt in proving that it works due to the nature of software and how it is used. This is the same as in the field of scientific research where someone comes up with a theory based upon their experiments and then their peers using a fair bit of skepticism try to prove the theory wrong. This is the crux of any scientific based research method. It is not about producing your theory it is about applying critical thinking to your theory as to why it can be wrong. Testing software is similar in its approach it is very difficult to inform someone, who matters that the software is working. We can provide useful information about its behavior and what it actually doing

As a tester you need to apply critical thinking to your testing and to the evidence you produce. This is where being able to look at what you are doing and what information you find will a fair degree of skeptical thought. The scientific method is a useful skeptic tool to apply to your testing and to any other information that people provide to you as 'fact'. One approach is the use of FiLCHers:

Falsifiability	It must be possible to conceive of evidence that would prove the claim false.
Logic	Any argument in support of a claim must be both valid and sound.
Comprehensiveness	The evidence must be exhaustive—that is, all of the available evidence must be considered.
Honesty	The evidence must be evaluated without self-deception.
Replicability	If the evidence for a claim is based upon an experimental result, or if the evidence offered in support of a claim could logically be explained as coincidental, then the result must be repeated in subsequent experiments or trials.
Sufficiency	The evidence offered in support of a claim must be adequate to establish the truth of that claim.

If you look at each of these headings you can see that it is about trying to find evidence where your theory and experiment could be wrong, acting as a skeptic.

This is a vital skill for testers to possess to ensure that their testing is unbiased and factually correct. If you feel the skeptic is lacking then look at ways you can improve it.

Here are some suggestions to help you become a better skeptic:

- Learn about logical arguments and fallacies
 - This page has a substantial list of them
- http://changingminds.org/disciplines/argument/fallacies/fallacies_alpha.htm
 - This book is a wonderful illustrated book proving examples of fallacies and bad arguments
- <http://www.amazon.co.uk/Illustrated-Book-Bad-Arguments/dp/1615192255/>
- Practice using skepticism – apply it to your work and the work of others.
- Use James Bach's 'Huh, Really, So' technique
- <http://itknowledgeexchange.techtarget.com/software-quality/james-bach-on-critical-thinking-huh-really-so/>
- Use reflection to help improve your skeptic skills.



John is tester, blogger, tweeter and author who has a passion for the software testing profession. He is keen to see what can be of benefit to software testing from outside the traditional channels and likes to explore different domains and see if there is anything that can be of value to testing. At the same time he likes to understand the connections between other crafts such as anthropology, ethnographic research, design thinking and cognitive science and software testing. He is currently writing a book on this called "The Psychology of software Testing".

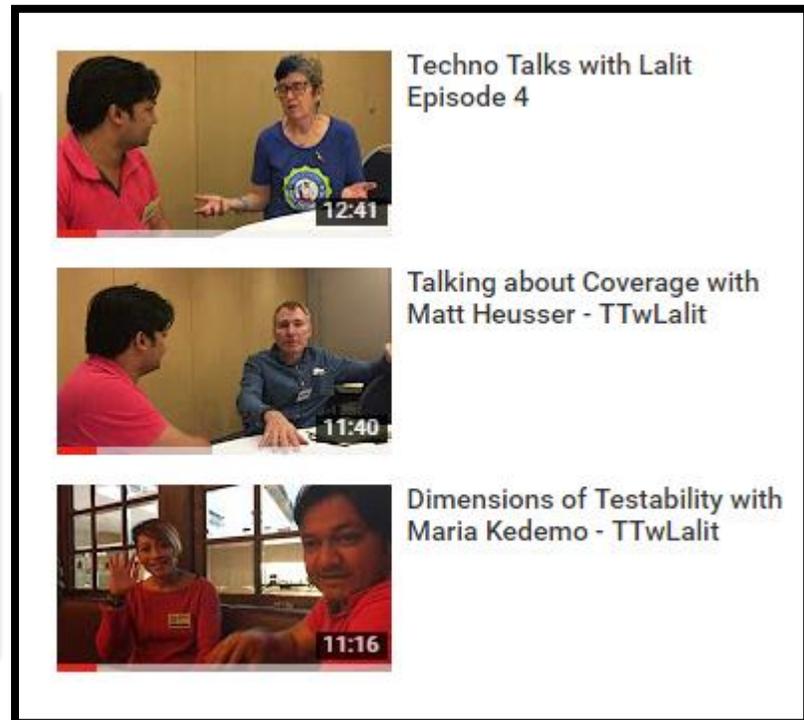
John has presented workshops and presentations at various events such as Agile Alliance, CAST, Testbash and Let's Test.

- See my previous post on reflection
- <http://www.steevo1967.blogspot.com/2015/10/testing-skills-7-reflection.html>

If you have some suggestions of your own to help others in their journey to being a great skeptic please let me know.

[Back To Index](#) 

Agile Testing Days 2016 Special episodes of [Techno-talks with Lalit](#)



Love to
Write?



Write For Us

Your ideas, your voice. Now it's your chance to be heard!

Send your articles to editor@teatimewithtesters.com



In the school of Testing

for your better learning & sharing experience



Data Migration Optimized QA Approach for a Multi-region Roll Out

- by Alex Placid and Sushma Suresh

Data Migration is one of the most puzzling initiatives in the current world of Information Technology as most of the current projects are working to upgrade their technical stack or move to better technology to resolve on performance issues. Though migration seem to be one causal word, the existence of these projects yield high very business benefits but has its own risk as it deals with volume and complexity of data.

Many of the projects involving migrations are the replacement of the existing systems with existing data. In all of these projects we end up creating data migration process or data conversion cod, to move the data from legacy systems to the new systems. As all these systems are currently used by the end users which are live in production, the conversions of the data really matters as the future business/functionality depends on the data being logically equivalent to the legacy system.

In order to reduce risk and work in optimized approach, QA migration approach and strategy plays a major role to ensure that the migrated data has been placed with appropriate transformation rules

What kinds of tools are available in the market?

Most of the Banking and Insurance applications were developed in mainframes and Visual Basic few decades ago. Now these applications are getting a new look in the current industry due to infrastructure, increase in DB, ease in use of application and so on; Hence Migration has become the KEY word in today's industry. To make these achievable, there are lots of Enterprise Transformation Logic tools available in the market for Data Migration load and validation.

Organizations based on their requisite, have specific kinds of tools or framework developed for their use. Most of the organizations prefer their own in-house developed tools as they provide more flexibility to customize, maintenance and are much cost beneficial too.

Validation approach for an Optimized multi-region roll out

In today's world with large Insurance and Banking operations are spanned across different regions that address their work model in various countries. As these organizations migrate to a model where business processes, infrastructure and resources can be standardized and reused across various discipline. Below are few of the approaches that was charted for one of the Strategic Insurance Migration Program:

Data Quality Check

- **Business Scenarios based Check** to ensure that the data has been correctly migrated as per mapping rules given from the business. These are scenarios are handled in the migration which are primarily driven by the mapping document (Source to Target) to ensure all the transformation logic are applied. The focus should be primitively on testing every rule / transformation logic specified in mapping for individual fields
- **Integrity Check** to ensure the source data are in sync with the target data and data integrity is verified. This ensures that mapping rules which are based on certain assumptions/constraints are working as expected. This aids in cleansing bad / poor quality data issues
- **Reconciliation Check** to ensure whether the count of data element available in source tables matches with the data available in the target tables removing all the data quality issues as addresses in the functional specifications. This is a high level dashboard indicating the separate reasons for dropping various data elements and list of dropped records. Once this report is available, the dropped data elements can then be reconciled with the load balancing reports.
- **Data Reports** summarize the post migrated data element records which helps the Operation team to aggregate the data between the source and target application

Functional Quality Check

Data Migration Automation is one of the strategic requisite that has been looked upon due to large migration engagements popping up through. Though Automation could save time and effort validating the data between the source and target most of the time, this does not give a full proof to the testing. The nitty-gritty is sometime been overlooked and hence functional validation becomes the need of the hour.

Functional validations give confidence to the Business users to check how the migrated data processes on the newly built system. Sometime during the requirement harvesting stage, the Sponsor/ Business Analyst would have not envisaged on how some of the data would be available in the newly built system and hence there comes issues if testing is not approached in the right manner.

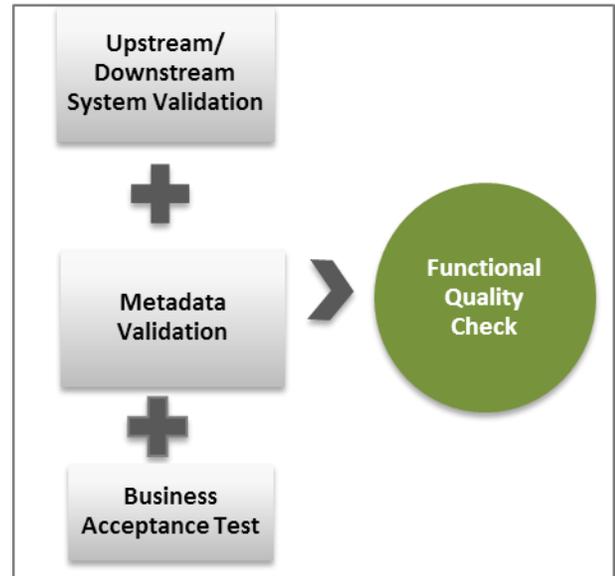
When we perform similar kind of validations for regional roll outs, though it is a repetitive process the data identifications plays a vital role. There might be millions of data being migrated as part of the project and it is essential to choose the right kind of combination to perform the validation.

The validation should be performed on the below key factors

- Upstream/Downstream System Validation
- Metadata Validation
- Business Acceptance Test

Performance Quality Check

There is a need to check if the migrated data can be scalable and how the system performs with the new system using the migrated data. There are multiple technical ways to expedite the validation of the execution timeline. Database performance, Query analyzer, SQL Profil-er, Data Stats, Indexing Rate , Memory Cache, Multithreading, Crawling etc. are different technical ways to help Data migration validation process to be done in the within the agreed timeframe.



In addition to the various measures, performance check on the system using the migrated data should also be measured. This would give assertiveness for the Business team on the migration process

- **Performance to be conducted on selected** data sets based on their criticality and other factors. This has to be conducted during the Non business hours to check the time in-volved in getting the data retrieved
- **Performance to be conducted on sample random** data sets that can be decided based on the requirement of data sets and other factors. This has to be conducted during the business hours to validate the real-time scenario

References:

- Practical Data Migration by Johnny Moris – Edition 2012
- DS8870 Data Migration Techniques By Bertrand Dufrasne, Alexander Warmuth, Jo-achim Appel, Werner Bauer, Susan Douglass, Peter Klee, Miroslaw Pura, Mark Wells, Bjoern Wesselbaum; Copyright by IBM Redbooks
- [ERP Model for a multi-Region Rollout](#) - Copyrights 2013 - Accenture



Alex Thuruthal Placid has been working as Test Manager in Cognizant Technology Solutions, UK. He has over 12 years of experience in managing Strategic testing engagements. He has played vital role in providing test management solutions for critical programs. He has orchestrated software test processes for end to end test management solutions.



Sushma Suresh has been working as Project Manager in Cognizant Technology Solutions, Australia. She has 10 years of experience in managing enterprise IT project lifecycle through all testing phases. Her test approaches has aided in resolving problem statements for critical projects. Sushma completed her master's degree from Anna University (India) and holds Prince2 & Six Sigma Certifications well.



Outdated Testing Concepts #3

- by Viktor Slavchev

I was having this internal struggle for a long time – certification. At first, I thought “it will be cool to be certified, it will probably drag some attention to me”, keeping in mind I was green and didn’t have enough routine. Then, I thought that certification is a must, I should have it in my skill set at any cost, and it will show people that I spent some time and invest in my own development. Now, I believe **it is up to me**, to be recognized as a skillful tester and **it has nothing to do with being certified**. This week’s outdated testing concept is dedicated to certificates and other cool scratch paper.

Outdated testing concept #3: Certified means qualified.

I have been thinking on this article for quite some time, I even failed to post it last week as I had doubts on talking about it. I was having this struggle within, whether or not I am qualified enough to give my opinion on certification. And a miracle happened, an argument in the local QA Facebook group I participate in made my belief solid, that there’s something fishy in certification, after all.

What was certification supposed to mean?

I will try to speak on testing certification only, but I think this **applies to all certificates**. Many people have different expectations towards software testing certification, unfortunately most of them wrong:

Certification is supposed to mean you took some formal software testing education

This is a broad topic and I will only scratch the surface with what I have to say, but software testing **could not be learned by notebooks**, nor certification courses. It is a **practical activity** and could be learned, educated and trained only through **practice**. You can learn the **principles of testing**, you can learn the **testing terminology** or the **testing glossary**, you can learn common techniques that are used in testing, but you can’t learn how to perform testing at an expert level. If you want to go deeper in the topic – education isn’t what it was supposed to be, we are aware already that **great minds are not “laboratory grown”**, e.g. could not be cultivated in schools, colleges and universities. I am not saying schools and universities are useless, I’m saying **they are not enough**. Formal education isn’t satisfying our needs for natural interest, our professional choice, etc. I referred many times to [Sir Ken Robinson](#)’s work, in one of his talks [“Ken Robinson: How to escape education’s death valley”](#), he is talking how modern education is applying the so-called “fast food” approach towards

students. This is commonly observed in testing certification as well, the “**one size fits all**” concept that tried to make you believe you can **learn something easily and by following simple steps**, which is far from true. Another important point that Sir Ken Robinson makes, is that above all, **intelligence is diverse**, and we should take this diversity into account, while educating, in software testing or out of it.

Certification is supposed to standardize testing.

This is another fallacy that has to be brought to rest. The purpose of certification is not standardize the process, but to assert that you gained specific expertise. In fact, there is a way to standardize the process already like the [IEEE standard for test documentation](#) and if you take a look at it you will find out it is not that **cool as it seems**. If we have to follow the standard point by point, the testing process will become unnecessarily document heavy, slow and boring, resembling more of a court case rather than experimental process.

I will find better job / get promoted, if I am certified.

It is possible, certificates will have certain value for your future or current employer, but that shouldn't be all. Me personally, I **wouldn't trust employer who uses certificate only to evaluate an employee**. Most of the times, when it comes to recruitment, your motivation and willing to progress will play more general role in decision, rather than the certificate.

You can go and continue the list. What probably pops in your mind as a question is..

Who values certification, anyway?

My personal opinion is we can split this group in 3 sub-groups:

1. **New testers who are looking for a way to prove themselves** – and this is kind of normal. Every new tester is trying to prove his/her value, wants to **progress** fast and dynamically and most of all is not familiar with all testing fallacies, one of which is the certification.
2. **Testing professionals who invested too much in certification, themselves** – psychology is a weird thing and it states that, when an individual **makes a reasonable choice**, he or she **must find a way to justify it**. And this is the case with this sub-group of testers – they spent probably \$200 for the foundation level course and probably much more for the intermediate, manager and ultra-super-mega-uber-testing-ninja-master-rockstar level. It is **understandable** why they will tell you certificates are valuable, otherwise these guys would have to **admit that they threw their money into the ocean**.
3. **Non-testing professionals who are involved in recruitment process** – and we have to make a condition here, **even when they are interested in you having certificate**, don't be too proud with it, it is just **conversation starter**, you will have to actually prove what you can and not just **rest on “certification laurels”**. Why are HRs and recruiters interested in certification? Well, normally they have none to minimum knowledge on what a really good testing professional is, so if he has a certificate, it is a really comfortable social contract. If he happens to be

a **complete moron**, knowing nothing about testing, they will blame the bastards that certified him, because it's **their responsibility** to tell good testers from the incapable ones. And if he turns to be greatest professionals – "Yay, see that? I told you, certification is an important thing to have for a tester."

So, what is the truth about certification?

The truth is – **it is useful to some extent**. The reason why people are basically impressed by certificate is – it shows that you have the **dedication** and the **motivation to invest in your own professional development**. It is also useful, or at-least it is foundation level, because you can get **familiar** with some **basic terminology** in the testing domain. That doesn't necessarily mean you will be able to explain what it means or understand it profoundly, but you will be **comfortable using it** in certain occasions.

Should I get certified, after all?

I am not the one to answer this question, it's you and only you. My suggestion is, if you want to invest in yourself and your professional development, invest in **practical courses** and not in certification. Testing is a practical activity and it is **only learned and explored effectively through practice**. And not testing only, you can invest in learning some basic programming or networking or any other generic IT knowledge. That will always be in your favor. I don't want you to stay off certification, because of me, I don't want you to take it because of me, either. It is all up to you and your personal philosophy. **My own opinion is** – if you are **passionate** about your profession, if you **strive to progress**, if you make yourself **stand from the crowd** by being proactive or a blogger or a conference speaker, I believe no one will ever ask you if you have a certificate.

And one more suggestion if you decide to take certification, anyway. It is from the book "**Lessons learned in software testing**" by Kaner, Bach and Pettichord – I quote by memory "**if you can get a black belt for two weeks only, try to stay out of fights**".

That's it for this time, thanks for reading and as always I would love to read your opinion on the topic. That was it for outdated testing concepts, as well. I think I said what I had for now, on this topic.

Someday I will probably continue with a new one. Thanks, good luck!



Viktor Slavchev - Senior QA @ siteground.com, from Sofia, Bulgaria.

Currently involved in the area of web hosting, Viktor has previous experience and interests in the area of mobile and web testing, non-functional testing and in the telco area.

Viktor is a passionate tester, who doesn't believe in "golden rules" about testing and thinks everything has to be put to a test. He likes to explore new trends in testing and learn from them. Strong believer that testing isn't simply a technical, nor soft skill, but can benefit from a lot of humanitarian disciplines like psychology, philosophy (epistemology in particular), sociology, anthropology, history and others.

Viktor is also passionate in teaching testing and helping new testers in mastering the craft, he's currently giving lectures in an a local IT academy, called Pragmatic, on various testing topics like Exploratory testing, Mobile testing and Non-functional testing. In his spare time, he's a passionate MMORPG gamer, listens to loud rock music and enjoys reading interesting books. You can read more from Viktor at his blog - mrslavchev.com or follow him on twitter @Mr_Slavchev

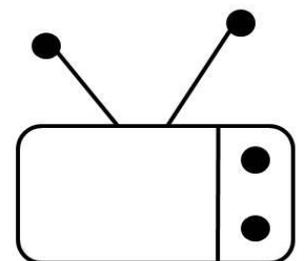
TECHNO
TALKS
WITH LALIT

BRAND NEW SHOW: Techno Talks with Lalit on www.tvfortesters.com

Tune in now

[Checkout all Episodes](#)

TV for Testers



Your one stop shop for all software testing videos



Sharing is caring! Don't be selfish 😊

[Share](#) this issue with your friends and colleagues!



Happiness is....

Taking a break and reading about **testing!!!**



Like our FACEBOOK page for more of such happiness

<https://www.facebook.com/TtimewidTesters>

T ' Talks

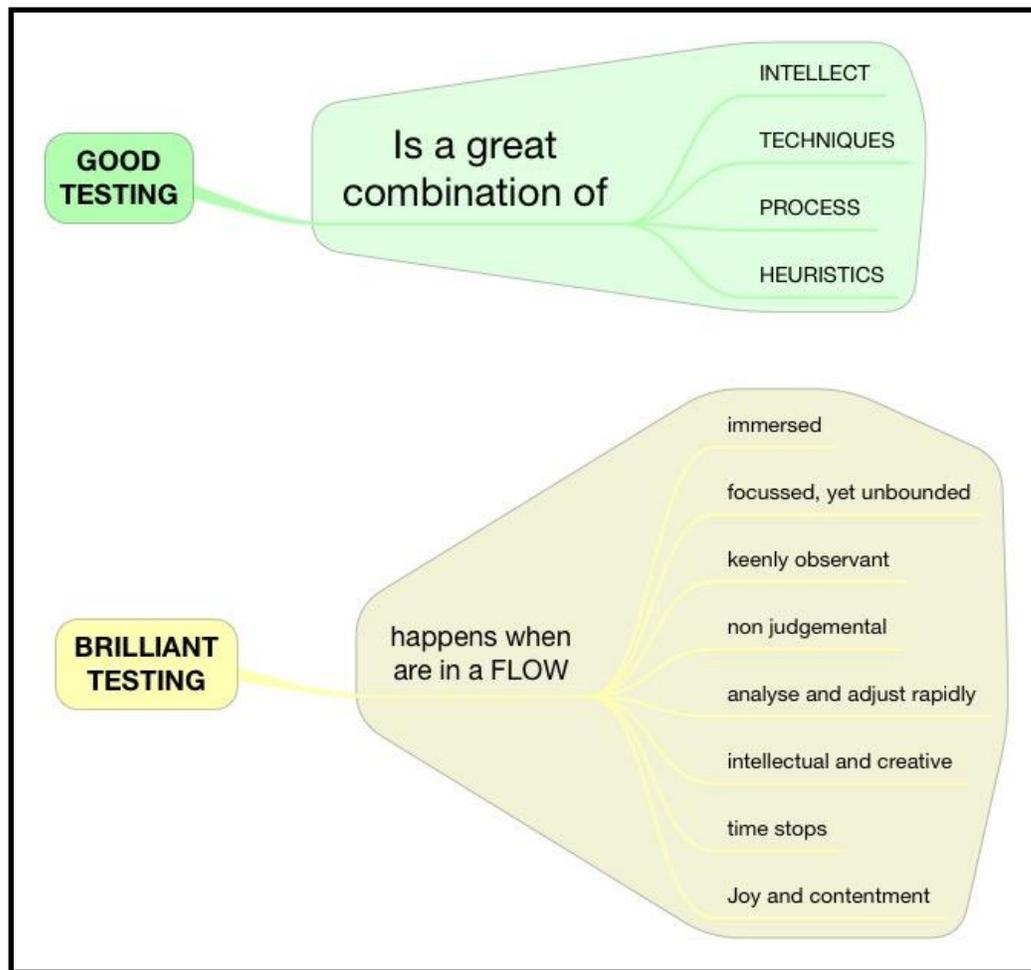


T. Ashok exclusively on software testing

Be in a FLOW. Test BRILLIANTLY

As engineers we use good techniques, tools, process and intellect to do things well. So how can we do better, rather brilliantly? Having possibly exhausted all things "external", in terms of tools, techniques, process and intellect, it is time we considered the huge "internal" potential that we all possess.

It is about going beyond the intellectual mind, deeper into the sub-conscious to harness the infinite potential. A good start for this would be get into a state of 'flow'.



So, what is FLOW?

Flow is the state when you are immersed in what you are doing, when you are truly mindful. It is when all energies are fluidly harnessed, when you do brilliant work without tiring or trying hard, when you are very observant dispassionately, when you are able to fine tune actions with extreme agility, when it seems that time has frozen i.e. you do enormous amount of work in the given time and it seems that you are not doing any work and experience joyful contentment.

So how can we get into the state of flow?

When multiple sensory excitations converge harmoniously, there is a good chance of entering that state of flow. What does this mean? It is engaging the various senses well - colours, picture, visual test, mind maps for the eyes, using pen/pencil, paper for the touch, and maybe background music or talking to oneself for 'engaged HEAR'. Note the keyword is 'harmonious'.

Engaged DOING is key to unleashing your power!



When we stimulate our creative side with interesting visuals, tactile and possibly sound, it enables the creative side to be activated enabling us to get into a 'engaged state', a state of mindfulness.

Exploit visual thinking by using visual text , sketch maps, mind maps to : (1) enable deeper understand the system under test (2) to sketch test strategy (3) jot test ideas (4) note down observations, questions, suggestions (5) list scenarios (6) record issues.

Exploit the tactile sense by using pen/pencil on paper or finger/stylus on tablet instead of keyboard. The objective is to be write/draw freely rather than be constrained by the keyboard.

If you want to engage your auditory sense, quiet talking to yourself, melodious quiet whistling or if you are music person, then a suitable background song played could enable you to get into the flow.

To ensure that we can perform in the state of peak performance in the FLOW state, it is imperative that we do testing in short sessions. A session may be anywhere between 45-90 minutes. The key is to stay focused, setup an objective at the start of the session and then engage as stated earlier to get into a state of FLOW.

How does this help?

When we are fully engaged, in a state of FLOW, it is no more about just the left brain centred logical/deductive thinking, it is a brilliant combination of logical thinking with the harmonies stimuli resulting in a creative non-judgmental expansive multi-dimensional thinking. This is when we exploit the infinite power of what we possess, delving into the sub-conscious that is far bigger than the conscious mind. Interesting questions pop up, ideas germinate rapidly, actions are done quickly, and smallest observations captured resulting in brilliant effective testing.

I have been working on incorporating these ideas into a style of testing that I call “Immersive Session Testing (IST)”. This uses paper based ‘workspaces’ to encourage tactile feel, incorporates good metaphors to enable great visual thinking, and is powered by a set of techniques, heuristics/guidelines to ensure a scientific approach to testing.

In closing...

Test automation allows us to do more, and machine intelligence allows us to do better. It is now necessary for us to delve deeper so that we complement the machine rather than compete allowing us to be super-efficient by being smarter, and this is where the state of FLOW to exploit the power of sub conscious could be very handy.



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





T esting is not a Career

It's a journey...

If you talk to a number of successful testers today you will see a common thread. The majority of them did not plan to become a tester, they [became testers almost by accident](#) via a number of different paths.

If so then, which is the best path to becoming a successful and professional tester?

The truth is, I don't know.

Furthermore, I am not even sure there is a correct path.



But I understood something more interesting from talking too many of these successful testers in the last 20 years.

It took me a while to realize this, as I had to be able to ignore what they were doing and even what they were telling me, in order to be able to focus on who these people were and what set them apart...

[Successful testers have the following Qualities that I have found in almost all of them:](#)

1. **Empathy** – the ability to see the world through the eyes of others and understand what is important to them.

2. **Curiosity** – to be able to dig deeper into the real reasons behind behaviors that others would simply dismiss as not interesting or irrelevant.
3. **Technical Skills** – to go deep into layers of their products that are usually not explored or exploited by other testers.
4. **Self-Learners** – to make sure they don't wait for information and knowledge to reach them, they are looking for ways to expand their horizons on their own.
5. **Stubbornness** – to not get NO for an answer when people are trying to tell them they are wasting their time.
6. **Communicative** – to make sure they are heard and understood when they need to communicate their ideas. I focused on this point exactly in a webinar earlier this year – [view webinar](#)
7. **Humility** – to understand when they are pursuing the wrong path and make corrections without drowning on their egos.
8. **Perseverance** – to understand results do not come from quick winning sprints but from running the long marathon races.

I know there are many lists and many people writing them, but I am keeping this one handy and the next time someone asks me how can they become better testers I will start by ask them to look into the list and really asking themselves if they have at least most of these points.

If they don't see themselves there, then I will suggest they look for another profession.

And if they do, I will ask them to look for ways for using these skills in order to make their work professional. Trusted that when done correctly and smart enough they will be able to become great testers in their own time.



Joel Montvelisky is a tester and test manager with over 15 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.

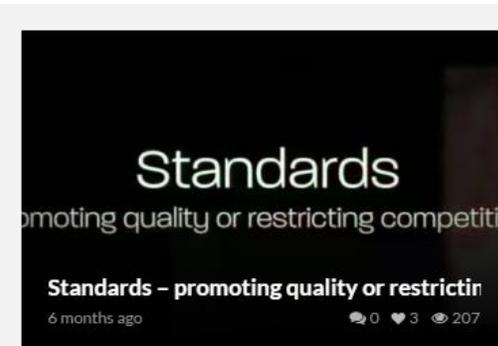
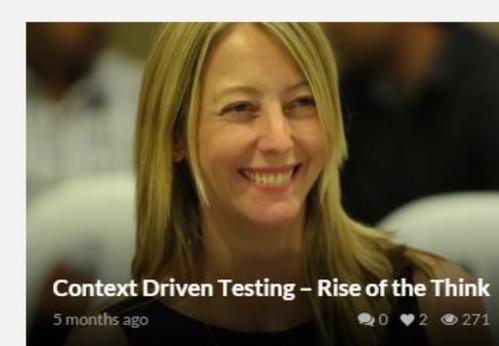
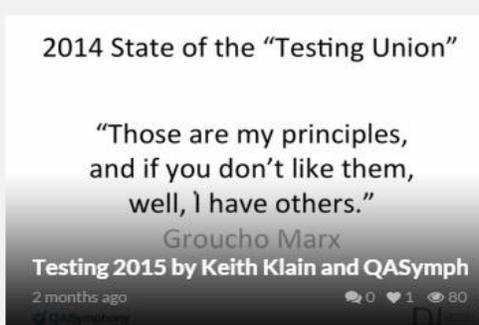
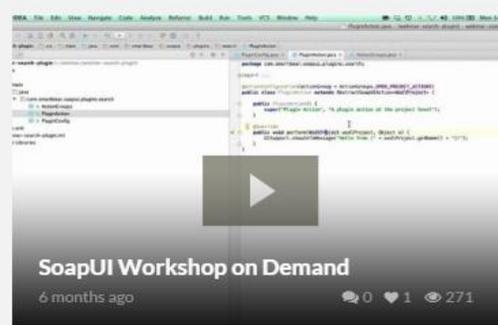
He also imparts short training and consulting sessions, and is one of the chief editors of ThinkTesting - a Hebrew Testing Magazine.

Joel publishes a blog under - <http://qablog.practitest.com> and regularly tweets as [joelmonte](#)

Got tired of reading? No problem! Start watching awesome testing videos...

TV for Testers

Your one stop shop for all software testing videos





www.talesoftesting.com

What does it take to produce monthly issues of a most read testing magazine?
What makes those interviews and articles a special choice of our editor?
Some stories are not often talked about...otherwise....! Visit to find out about
everything that makes you curious about **Tea-time with Testers!**

Advertise with us

Connect with the audience that **MATTER!**



Adverts help mostly when they are noticed by **decision makers** in the industry.

Along with thousands of awesome testers, Tea-time with Testers is read and contributed by Senior Test Managers, Delivery Heads, Programme Managers, Global Heads, CEOs, CTOs, Solution Architects and Test Consultants.

Want to know what people holding above positions have to say about us?

Well, hear directly from them.

And the **Good News** is...

Now we have some more awesome offerings at pretty affordable prices.

Contact us at sales@teatimewithtesters.com to know more.





Every Tester

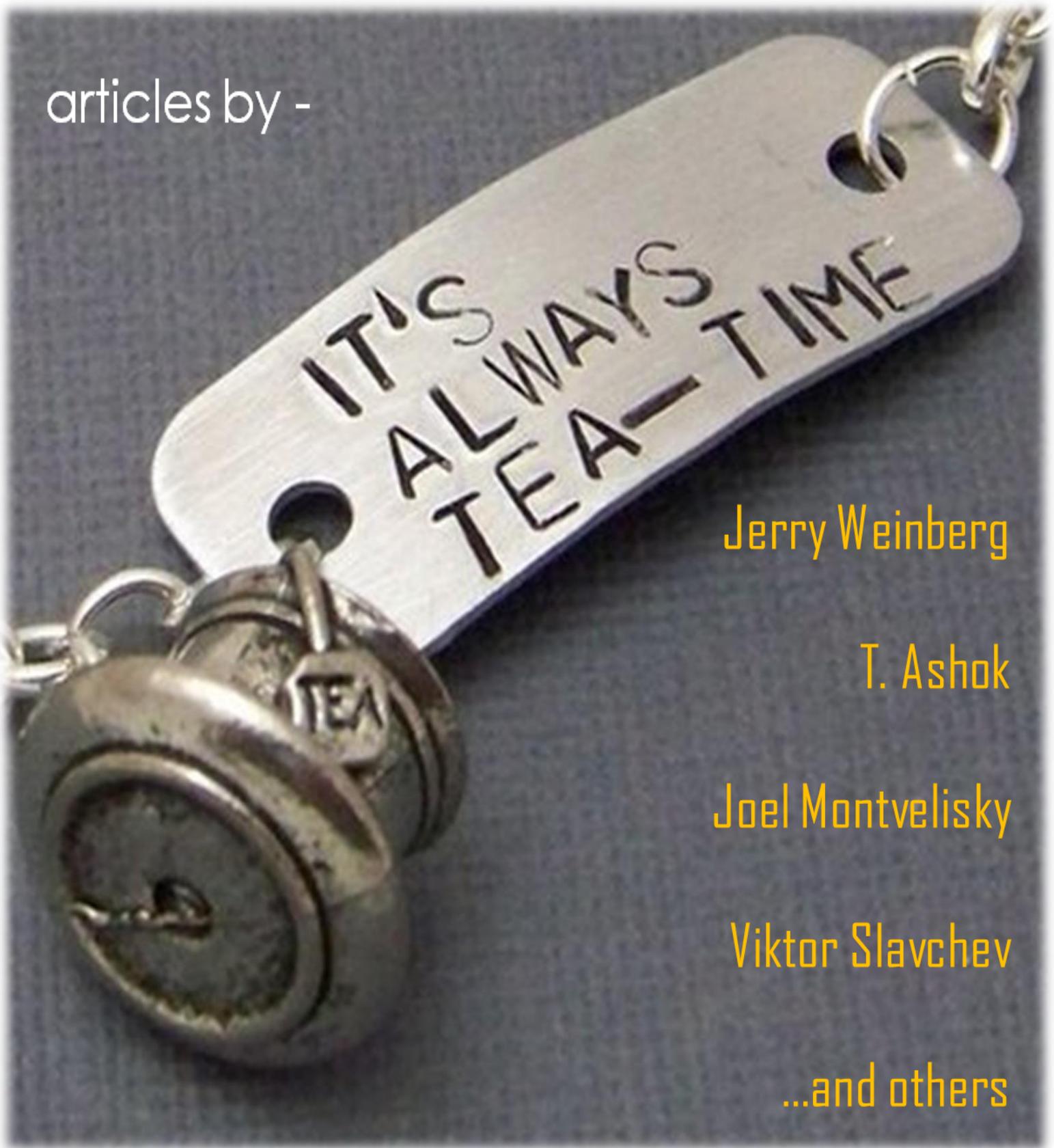
who reads **Tea-time with Testers**,

Recommends it to friends and
colleagues .

What About You ?

in ne>xt issue

articles by -



IT'S
ALWAYS
TEA-TIME

Jerry Weinberg

T. Ashok

Joel Montvelisky

Viktor Slavchev

...and others

our family

Founder & Editor:

Lalitkumar Bhamare (Pune, India)

Pratikkumar Patel (Mumbai, India)



Lalitkumar



Pratikkumar

Contribution and Guidance:

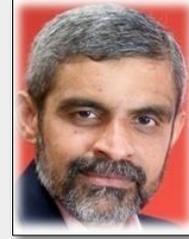
Jerry Weinberg (U.S.A.)

T Ashok (India)

Joel Montvelisky (Israel)



Jerry



T Ashok



Joel

Editorial | Magazine Design | Logo Design | Web Design:

Lalitkumar Bhamare

Cover page image – Metaphrasi

Core Team:

Dr.Meeta Prakash (Bangalore, India)

Dirk Meißner (Hamburg, Germany)



Dr. Meeta Prakash



Dirk Meißner

Online Collaboration:

Shweta Daiv (Pune, India)



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 a **FREE** copy,
Subscribe to mailing list.

SUBSCRIBE

Join our community on

facebook.

Follow us on - @TtimewidTesters



www.teatimewithtesters.com

