THE INTERNATIONAL MONTHLY FOR NEXT GENERATION TESTERS

Tea-time with Testers

OCTOBER 2013 | YEAR 3 ISSUE IX

Jerry Weinberg
Managing Others – The Manager's Job (part 3)

Naomi Karten
Introverts working with Extraverts

Lorinda Brandon
The STEM crisis: Fact or Fiction?

Jim Holmes
Why Automate?

Bernice Ruhland
Introducing change in your Testing Department

Sashidhar Penumala
Can we ship?

Over a Cup of Tea with T. Ashok

QAZone presents...

ThinkTest.

The Conference for Thinking Testers

December 14-15, 2013 | New Delhi - India

ThinkTest talks

Don't miss the chance to hear from legends like James Bach and other experts in industry...

JAMES BACH

MICHAEL LARSEN

SMITA MISHRA

LALIT BHAMARE

AJAY BALAMURUGADAS

MEETA PRAKASH

T ASHOK

MUKESH SHARMA

Click HERE for more details

TEA-TIME WITH TESTERS

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

Created and Published by:

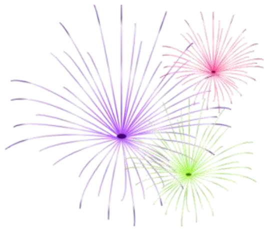
***Tea-time with Testers*.**
Hiranandani, Powai,
Mumbai -400076
Maharashtra, India.

Editorial and Advertising Enquiries:

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

Editorial

Seasons, Celebrations and Surprises!



So, the season of festivals is finally here!

By the time you read this issue, Diwali celebrations in India would have been already started. Diwali is festival of lights. This reminds me of *Tamaso ma jyotirgamaya,* a mantra from Upanishad which means, *lead me from darkness to light.*

On occasion of this festival, I want to wish our all readers a happy and prosperous Diwali. May each of us see light of wisdom that will eliminate the darkness of ignorance.

We have always offered special issues at special occasions and this season is not an exception. I am glad to announce some new article series by Jim Holmes and Bernice Ruhland starting from this month. I am sure you'll enjoy them. Special thanks to Naomi Karten for writing interesting article, especially for TTwT. And we have got more from T. Ashok apart from his T'talks column this time. Do not forget to read his interview.

And yes, we have not forgotten about Thanksgiving ☺. Mike Lyles and JeanAnn Harrison are working on some super-special themes and you'll get to read their articles starting from our Thanksgiving issue.

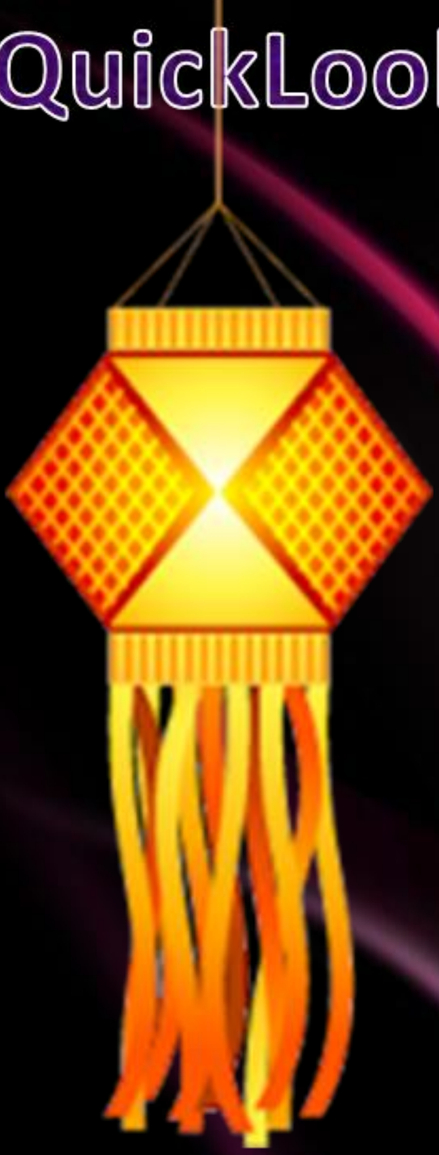
That's all from my end for now. I am sure you'll enjoy reading this edition Tea-time with Testers, as always.

We'll meet soon…

[illegible]

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

QuickLook



Editorial

What's making News?

Tea & Testing with Jerry Weinberg

Speaking Tester's Mind

Introverts working with Extraverts - 14

The STEM Crisis: Fact or Fiction? - 17

In the School of Testing

Why Automate? - 21

Introducing change in your Testing Department - 24

Can we Ship? - 29

T ' Talks

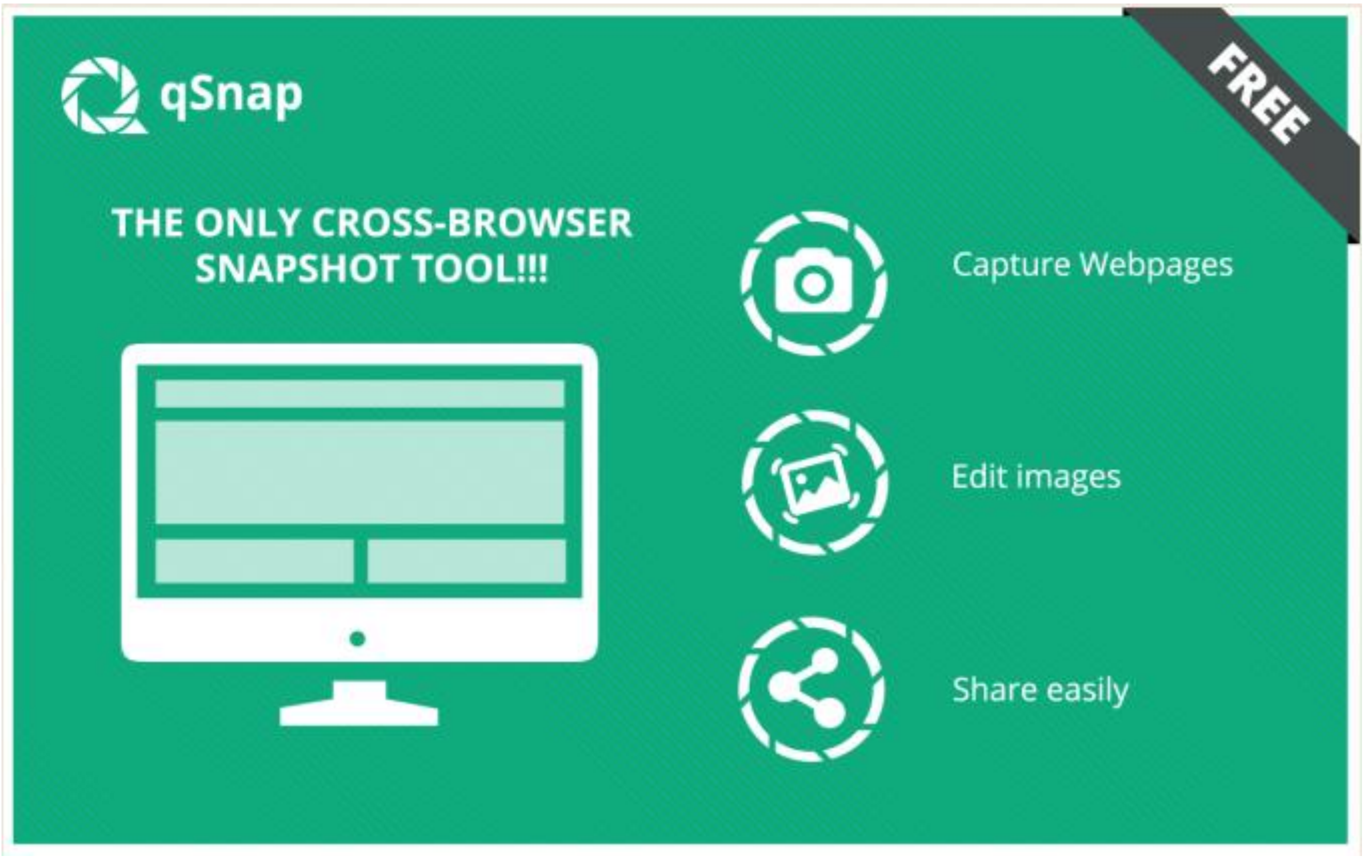
Lead and ye shall grow - 42

A cup of Tea with T. Ashok

Family de Tea-time with Testers

It's a Snap with QASymphony's Free qSnap Tool

Free multiple screen capture & documentation tool works across all browsers



QASymphony announced the launch of qSnap, a free screen capture and documentation tool and the only plug-in of its kind that works across all popular browsers. The tool can be downloaded at http://www.qasymphony.com/qsnap.html.

qSnap is not exclusive to QA testers or developers, but for anyone interested in using a screen capture tool that takes multiple screen shots at once and allows for jotting down notations. Screen shots can be shared, saved locally or stored on QASymphony's free hosting service.

Available as an unlimited free download, qSnap was made for anyone looking to quickly capture one or multiple screen images and share with others. As a documentation tool, qSnap allows users to add their own in-line annotations, note boxes and callouts with a single click. And with the free online hosting of the snaps taken, users can easily collaborate and share their documentation with others.

A browser extension, qSnap is available to the general public through the Google app marketplace and Google Chrome Store freely downloadable at http://dld.bz/cSfd8.

"Unlike other screen capture tools, qSnap is the only cross browser solution available that can easily capture one or multiple screens at a time," said Vu Lam, CEO of QASymphony. "This is not only useful to QA teams, but useful to anyone, such as marketers or trainers, who are looking to take screen captures, jot down notes, and seamlessly share them with anyone."

Most recently, QASymphony announced a JIRA connector for qTest, combining the powerful test management features of qTest with JIRA to create a complete QA management solution.

For more information visit: www.QASymphony.com

For more updates on Software Testing, visit → Quality Testing - Latest Software Testing News!

Your search ends HERE

Managing Others – The Manager's Job (Part 3)

There is no room in this novice manager's repertoire for coaching, teaching, or training. Contrast this view with that expressed by a leading management consultant:

The middle manager's job is to "grow people"—not to build a file of lifeless documents, or to spend half of each day in boring meetings. His role means walking around with an open mind toward listening and helping, and taking time to talk things over. It means being concerned about individual welfare and fostering the full potential of each person.

Even when you believe the One-Dimensional Selection Model, you still have to determine who is good and who is bad—the manager or the employee. As they say in the army, "There are no bad soldiers, only bad officers." An analogy would be someone trying to get attached to a network, but the network administrator was unable to achieve the attachment. If you were going to fire someone, would it be the person who wanted to be attached to the network, or would it be the administrator? So, if the manager is unable to get a member working well with the team, doesn't the model say you fire the manager?

Administering

As a manager, you are responsible for considering how much faster and better the job would have been done if you had done it yourself. You're also responsible for reflecting why, if you had done it yourself, the job would have been complete in 30 minutes. Also, you need to understand why it took you took three days trying to figure out why it took somebody else two weeks to do it incorrectly.

The incongruent manager never does figure out the reason for a late, incorrect project is poor management. Brooks's Law, of course, is the classic example of managers causing the very lateness for which they blame others:

Adding manpower to a late software project makes it later.

Congruent managers are not locked into blaming, partly because they know they are not passive victims doomed by Brooks's Law. Let's see how a congruent manager might deal with administering a late project. First of all, such a manager would understand the dynamics of the law, as shown in Figure A. The only feedback loop is through the manager's own actions, so that only self-blame would be appropriate. But self-blame is not necessary, because analyzing the diagram shows several ways to accelerate progress.

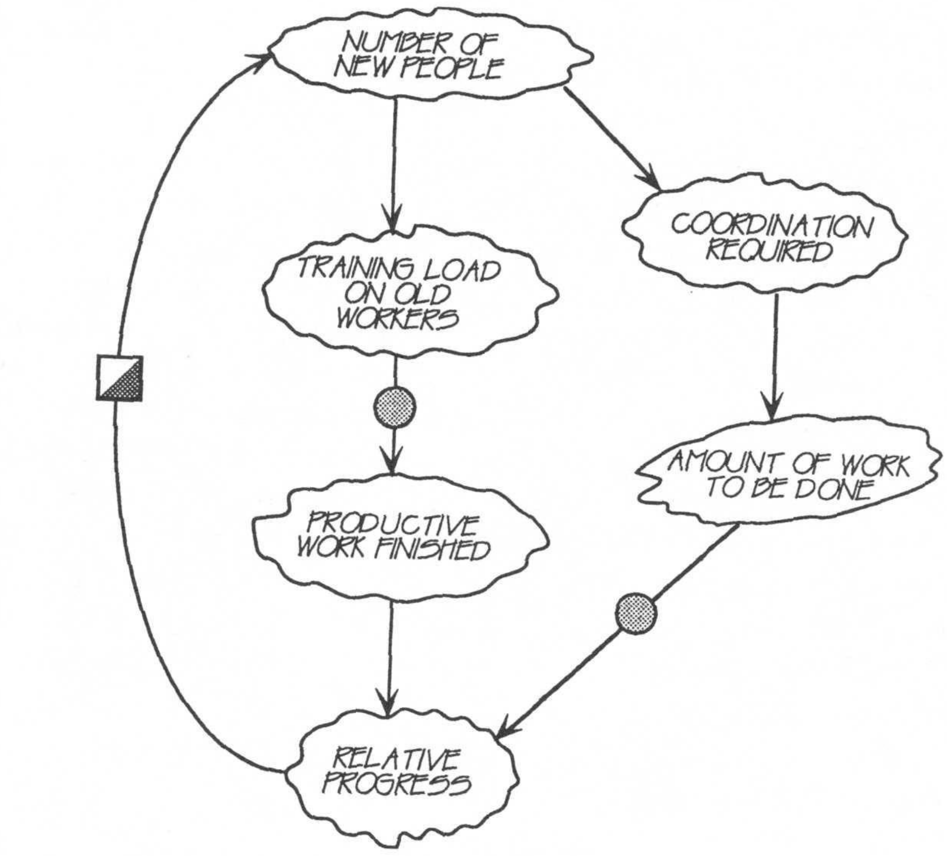


Figure A.

A manager, who understands the dynamics of Brooks's Law, understands what it takes to beat it. One observation is that although there are two paths by which adding people reduces relative progress, neither path contains a feedback loop. This makes it easier to beat the law, and the diagram suggests how. Based on the diagram, Brooks's Law could be refined as follows:

Assigning new people late in a software project to the tasks other people are already trying to do, makes the project later. So when managers want to put new people on a project or to assign established people to new tasks, they must assign them to *tasks not being done already*.

Because the project is in a schedule crisis, there will be plenty of such tasks lying around:

- locating faults behind lower-priority software trouble incidents (STIs)
- conducting additional independent technical reviews of code
- conducting additional independent technical reviews of test plans
- creating additional test cases
- fixing low-priority faults
- creating documentation
- updating documentation
- administering test runs when someone needs to be on-line

Of course, each of these tasks would be easier to do with the help of a more experienced person, but the new workers must be told they are to operate without such help, and why they are doing so. There are, however, a number of congruent steps a manager can take to improve the efficiency of their training:

- Give them a senior person as leader, taken from some area not in crisis. This gives them the advantage of experience with tools, languages, and systems, if not with the particular software in crisis.
- Give them a facilitating administrator who can get them resources and also give them the advantage of experience in dealing with the organization.
- Use burned-out experienced people who are no longer productive to work with them on specific understanding of the crisis software. A time-out of this kind can do double duty for the organization—teaching the new people and letting the burned-out old-timers have a taste of the newbies' enthusiasm.
- Schedule an occasional meeting—once a week for an hour, for example—with the experienced people. At the meeting, the new people can present prepared questions for immediate feedback. Knowing there will be such a meeting helps keep them from being desperate enough to break the rules and interrupt at other times.

This kind of sink-or-swim approach creates a stiff training regime, which some people may not be able to handle. The manager needs to monitor these people closely, to pull out those who cannot work in this environment, and to notice which ones have reached a level where they can make contributions to more critical problems. Not everyone will sink, and the manager will want to put the swimmers to work as quickly as possible. Unfortunately, managers may know who can handle the load emotionally, but they may not be able to tell who can handle it technically.

A good way to test people who may have reached this level is as members of error-location teams or as members of regular technical reviews. Because of the open nature of the work in these cases, the newcomers can be observed in action by the more experienced technical people. If they are contributing, it will be obvious to everybody, and soon they will find themselves being asked to handle things themselves, with salutary effects on the project. The next volume of this series will be devoted to managing through teams.

What Congruent Managers Do

Rather than allow the outline of this chapter to be totally dominated by an incongruent view of the manager's job, let's explore the consequences of a different model. This model says that managers are leaders, and Leadership is the ability to create an environment in which everyone is empowered to contribute creatively to solving the problems.
Certainly the word "empowered" has been overused and debased, so I want to be specific about what this definition means. In this model, the manager's job may be evaluated by one and only one measure:

the success of the people being managed. Note carefully that this says "success of the people," not "success of the project." Success of the project may be a *part* of the success of the people, but lots of sickness, broken homes, and psychological burnouts are not my ideas of "success of the people."

Over the years, I have interviewed people on dozens of successful—and thus well-managed—projects. When I ask the project members how their managers contributed to their success (empowered them), here are the things they said most often:

Our managers contributed to our success by

- offering positive reinforcement
- giving precise and clear instructions, and always being willing to clarify when they're not clear
- not constraining workers any more than is essential
- letting people fully explore the possibilities
- simplifying tasks whenever possible, yet making sure the tasks aren't insultingly easy
- making the timeframes clear, and giving the reasoning behind them
- paying attention to people's skills
- balancing the workload among all employees
- ensuring there is some real part to play for everyone
- teaching how to be supportive by being supportive of employees, and of each other
- teaching how to trust by trusting each other and their customers
- remembering what it was like to be an employee and to be managed
- answering questions correctly and honestly, to build trust
- getting good consulting advice and using it
- creating a vision of the problem and communicating it clearly to everyone
- providing organizational guidance when employees need it
- setting things up so people can experience early success
- not asking people to do things they aren't able or willing to do
- creating an environment in which it's okay to have fun
- making their objectives clear at the beginning
- being available to workers and being generous with their time
- understanding and forgiving mistakes
- valuing creative approaches, even when they are different from what they had in mind
- not forcing people to be something they're not
- finding the resources employees genuinely need to do their jobs
- changing their plans to fit the experienced environment
- resisting the temptation to change the rules in the middle of the project, unless it is absolutely essential
- explaining the reasons when something had to change
- always being up front with employees, even when it's embarrassing
- genuinely wanting people to succeed
- oh, yes, and occasionally making good decisions about hiring and firing subordinates

Does this sound like an environment in which you would like to work? Does it sound like the manager you would like to be?

To be continued in next issue…

Back To Index

Biography

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

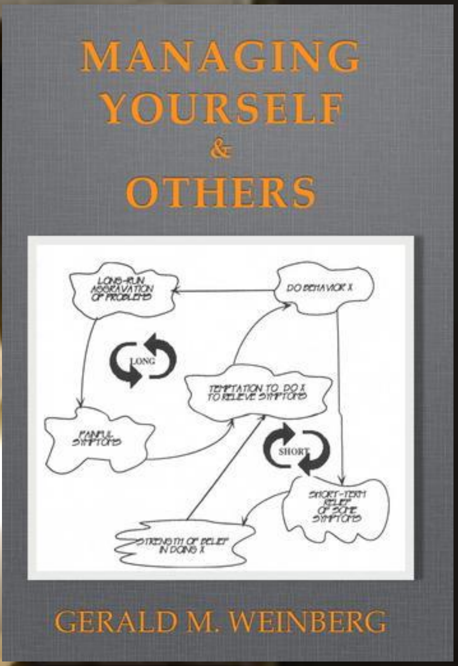
For more than 50 years, he has worked on transforming software organizations. He is author or co-author of many articles and books, including The Psychology of Computer Programming. His books cover all phases of the software life-cycle. They include Exploring Requirements, Rethinking Systems Analysis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

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

To know more about Gerald and his work, please visit his Official Website here .

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

MANAGING YOURSELF AND OTHERS is another famous book written by Jerry.

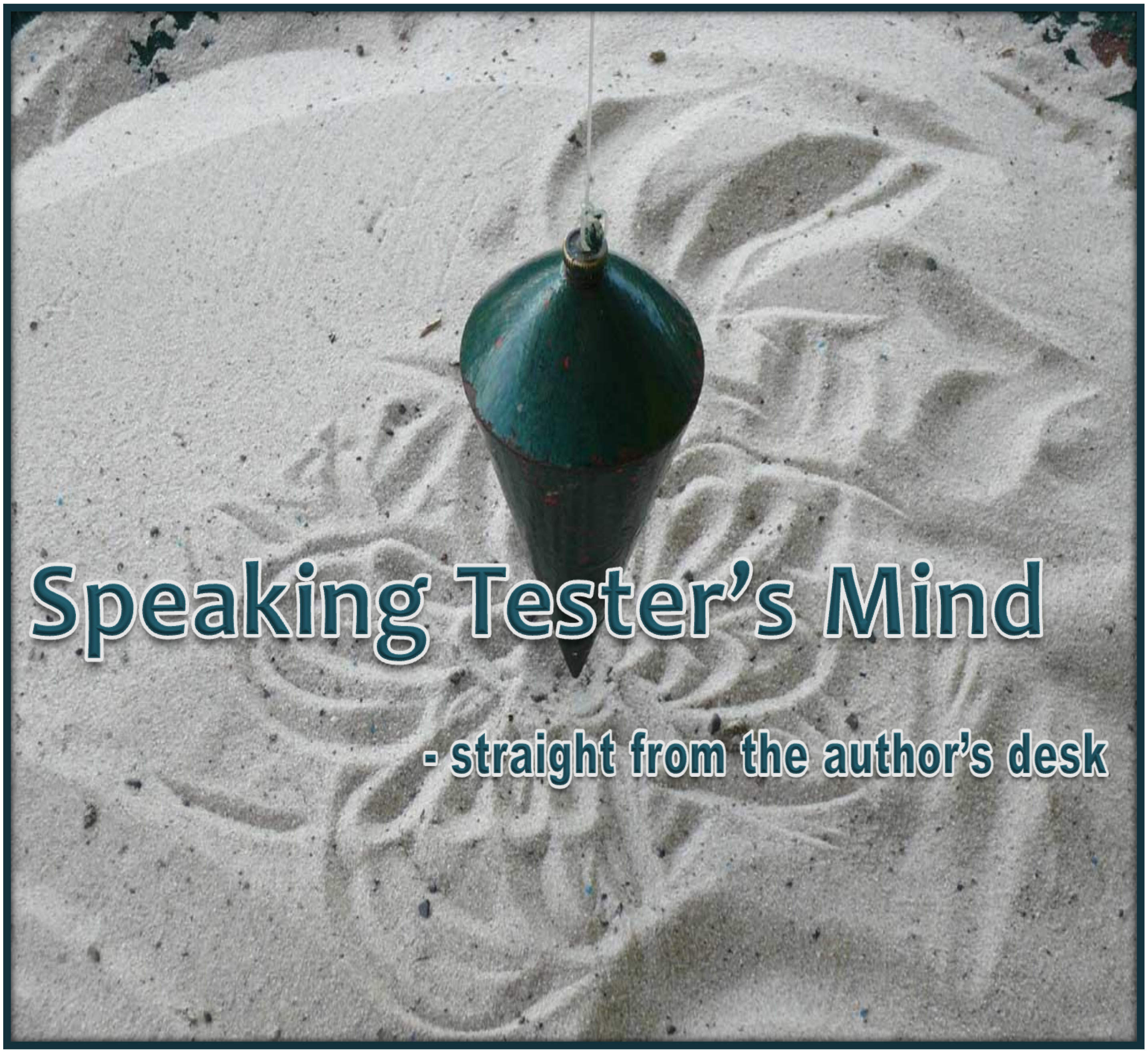


TTWT Rating: ★★★★★

Becoming an effective manager is the subject of this volume in Gerald M. Weinberg's highly acclaimed series, Quality Software. To be effective, managers must act congruently. Managers must not only understand the concepts of good software engineering, but also translate them into their own practices. Read this book to find out more.

Its sample can be read online here.

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


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

Introverts Working with Extraverts: The Challenges and the Benefits

by Naomi Karten

Some people insist I can't possibly be an introvert. They draw this conclusion because they see me engaged in a lively conversation or expounding on some point or giving a keynote presentation to an audience of hundreds. But none of these activities are contrary to what introverts can do.

I am, in fact, a lifelong introvert. I live much of my life inside my head, as so many introverts do. It's easy to find me at a party where I don't know anyone. I'm the one at the bookshelf scanning the books. And after being engaged in lively conversations or expounding on one or another point or giving keynote presentations, I'm likely to seek time alone so I can recharge.

It's only in the last several years that introversion has come into its own as a topic of discussion – and a topic viewed as worthy of discussion. You can find blogs galore on introversion, as well as numerous books and a steady stream of tweets and articles. Not so extraversion. There's relatively little written about extraversion outside of academic/research circles, although the increased focus on introversion has led to extraversion getting somewhat more attention.

Why is so much attention devoted to introversion and so relatively little to extraversion? The reason, I think, is that many introverts struggle to make their way in the world in a way that extraverts don't (or at least don't seem to). And that struggle can easily affect the quality and success of team efforts. Given that a test team – or any team, for that matter – is likely to have both introverts and extraverts on it, it's useful to reflect on the challenges introverts face in working with their extraverted colleagues, as well as the benefits.

Compared with extraverts, introverts tend to be more reserved, less expressive, and less animated. Introverts tend to process ideas internally before voicing them. Not all ideas, obviously, and not all the time, but often, we mull over thoughts and ideas before expressing them. We tend to be more comfortable interacting one-on-one and in small groups than in large groups.

Extraverts tend to be focused on the outer world of people and things. They thrive on interaction and, according to my extraverted colleagues, actually gain energy through interaction. They typically like to bounce ideas off their colleagues and figure things out through discussion. Introverts, by contrast, tend to be focused internally, to thrive on quiet, and to lose energy through interacting (or even, at times, just listening). And we're more likely to want to work out problems on our own before getting feedback from others.

(Note that in talking about introversion and extraversion, I deliberately use wording like "tends to" and "often." With this topic, as with most others that concern human behavior, it's a serious mistake to speak in absolutes, such as claiming that "introverts always…." or "all extraverts….")

It's been said that both extraverts and introverts have an inner life and an outer life: Extraverts have both on the outside; introverts have both on the inside. This is said somewhat in jest, but there's more than a grain or two of truth to it.

So what do introverts face on a test team in which some of their teammates are extraverts? Introverts may find themselves working with team members who talk at length (as some extraverts do), have boundless energy (as many extraverts seem to have), get energized by interaction (as so many extraverts do), and think out loud (as extraverts can do so well). Challenges, indeed, for many introverts.

But there are also huge upsides to working with extraverts. For one thing, they are comfortable in social situations (at least, that's how it appears to introverts), and that's something that can enhance collaboration both within and outside the team. Furthermore, their energy can be infectious. When I'm with extraverts, I become more extraverted because their energy invigorates me. I didn't fully realize how much I appreciated working with extraverts till the time I was on a team with only introverts. We got the job done but the difference in energy level was palpable. A few extraverts on a team can really perk things up.

In addition, extraverts' thinking out loud can generate ideas that we introverts may have also, but we have a tendency to mull over them – and perhaps edit them, revise them, modify them, rethink them, and edit them yet again – before we say anything (and sometimes, by that point, the conversation has moved on, so we say nothing). It's not that we're withholding ideas, as is sometimes mistakenly assumed, just that those ideas have to find their way from inside our heads to the outer world. Extraverts seem to have a direct connection between brain and mouth. Their ability to bounce from one idea to another can add possibilities to the team's efforts that might be overlooked otherwise.

If we introverts want to work effectively with the extraverts on our team, we have to take responsibility for what we need. For example, we can explain that sometimes we need time to take things in and reflect on them before responding. If someone asks a question or requests our opinion, we can ask if we can take a minute (or an hour) to respond. We can incorporate some quiet periods during face-to-face interaction and arrange some down time to recharge. And if extraverts go on at length so that our brains are about to burst, we can ask for a time-out. We can even do that *before* our brains are about to burst.

But it can't all be one-sided. If we want extraverts to accommodate our needs, we need to be willing to do the same for them. When extraverts are speaking, we can show some facial expression so they know there's someone home; seeing blank stares on our faces doesn't give extraverts the feedback many have told me they need. We can welcome approaches that extraverts thrive on, such as brainstorming – provided, of course, we balance them with some quieter approaches.

In addition, we can be more forthcoming in offering our ideas, so that we don't give the false impression that we're not team players or are not willing to do our part. We can learn to tolerate, and maybe even enjoy, the communication style that our extraverted coworkers may display, recognizing that in the midst of all that thinking out loud are great ideas that will further our efforts together.

Can introverts and extraverts drive each other crazy? Absolutely. Can we nevertheless work together effectively? Positively. And one of the best things we can do to improve our ability to work well together is to take time, early in a project or relationship, to better understand each other. In particular, we can:

- Explain and discuss our communication and work style as it relates to introversion and extraversion. This will go a long way in helping us understand and appreciate each other and the ways we're similar or different.

- Collaborate about how we can work together in a way that maintains respect for each other – and for ourselves. This may lead to each of us doing more of certain things and less of others in order to accommodate each other's needs and preferences, but it would not be surprising to discover that we all grow as a result.

- Give each other permission to raise concerns about how we're getting along as it pertains to our communication and work styles. Instead of letting frustrations fester, we can discuss what's working and what's not so that we can make adjustments – sooner rather than later – in support of our relationship, our priorities and our goals.

To help you get started with this sort of dialog, here's a self-assessment grid you may find helpful. Print out copies for each person on the team. After each person fills it out, compare your responses and discuss what the similarities and differences suggest for how you can work well together.

Or try this: in addition to filling out the form for yourself, fill out additional copies as you predict each of your teammates will respond. Then you can compare your prediction of their responses with their actual responses. While you're at it, print out some additional copies and compare responses with friends and family members. You'll learn some interesting and useful things about each other, guaranteed!

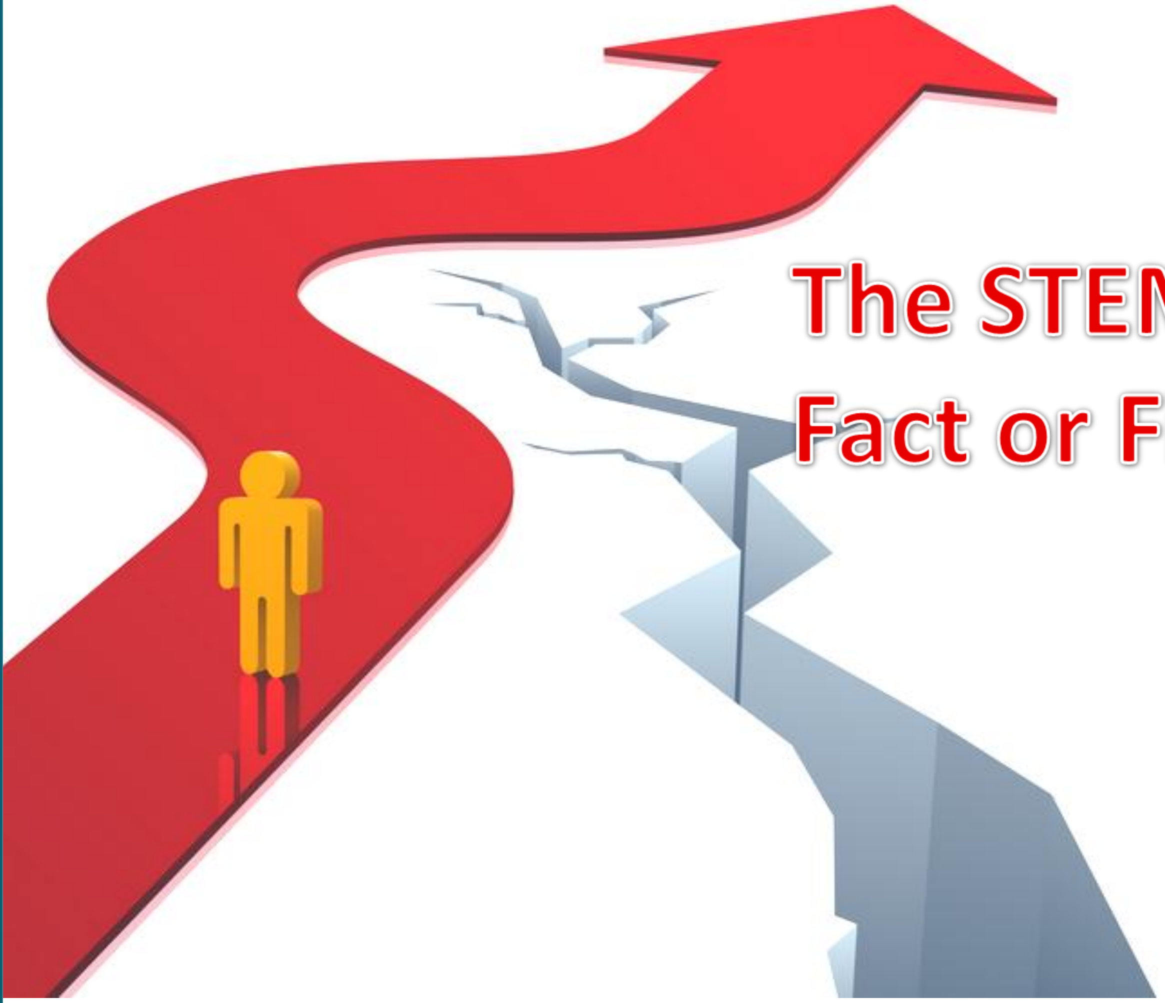
Being an introvert has not kept **Naomi Karten** (www.nkarten.com) from delivering seminars and presentations to more than 100,000 people internationally. Check her website for information on her guide: How to Survive, Excel and Advance as an Introvert.

Naomi has also published several books and ebooks, including Presentation Skills for Technical Professionals, Changing How You Manage and Communicate Change, and Communication Gaps and How to Close Them.

Readers have described Naomi's newsletter, Perceptions & Realities, as lively, informative and a breath of fresh air. Numerous issues of this newsletter are posted on her website. She writes articles on people and teams for TechWell. She tweets at @NaomiKarten.

She'd enjoy hearing from you (by email, of course) at naomi@nkarten.com.

Back To Index



The STEM Crisis – Fact or Fiction?

by Lorinda Brandon

When I began my career as a software tester in 1984, very few people in the software industry had computer science degrees. For most colleges, the degree itself had only been around for about 10 years. So we were forging our own paths out of pure interest and common sense. My memory of the time consists of a workplace that seemed pretty balanced with men and women. The industry itself was young enough that it hadn't sorted itself out on gender lines yet, and the gender ratio was never even a conversation we had. Granted, the workplace itself in every industry was heavily male as we were still at the turning point where women didn't have to choose between family and work but could actually have both, just like their male counterparts.

So, considering that it was perfectly feasible for large numbers of both men and women to build careers in software without the benefit of a computer science degree, why is it now being characterized as untenable? There's a dangerous conversation going on right now in this country. It's a conversation that implies a closed door in the technology sector if you don't have a STEM degree. While I don't disagree that we need more people in STEM if we are going to drive innovation for healthcare and environmental studies, I think the message that is beginning to infiltrate our society is not accurate for the software industry. And this message is more dangerous for women than it is for men. Women are naturally raised to expect closed doors at some point in their lives – you can love playing football in elementary school as much as the boy next door, but don't expect to go pro; you can be one of the best golfers in the world, but *you couldn't be a member of the top-ranked club in the world until 2012*; you can be president of your student class, but don't expect to be president of a company or the country any time soon. It becomes part of our wiring to enjoy what we love while we can but accept our fate when we see that "No Entry" sign.

So, when we say that an interest in STEM is a requirement if you want to have a career in software, we are edging the door closed for women. In part, that's because not as many girls (for whatever reason) are as interested in STEM as boys are. It seems as though *interest in STEM is waning across the board,* among both teenage boys and girls. However, the numbers are greater for girls. *According to this report,* "research shows that girls start losing interest in math and science during middle school. Girls are typically more interested in careers where they can help others and make the world a better place." What is particularly discouraging in this statement is that technology is not seen as being compatible with that career interest. In many ways, it is technology that will bring information to the isolated, connect food sources with the hungry, provide disaster assistance to the ravaged.

My question is this: at a stage when we are trying desperately to bring more women into technology, why would we start the conversation by preaching to 12-year-old girls that if they don't enjoy studying STEM, they cannot build a career in that sector? Why would we not approach the conversation from a place that resonates with their interests, like teaching them how to build a food pantry app or learning about Twitter's role in disaster communications? The reality about STEM education as a pre-requisite to a STEM career is that it is a myth. *According to a recent article,* "of the 7.6 million STEM workers counted by the Commerce Department, only 3.3 million possess STEM degrees." That's fewer than half. I have had a successful career in the software industry with a major in Art History. I know other men and women who have built long well-paying careers with non-STEM degrees, partly because we joined the industry when computer science was not a well-known degree choice. In fact, when there were no barriers to entry (like degree expectations), the *number of women in the field was higher than it is currently.* But with all the current noise about the need to encourage girls to stay with STEM education, the message that our daughters are receiving is that if they don't like math, they're doomed. If they don't want to learn to code, there is no place for them in software. Another closed door with a "No Entry" sign posted on it.

I would, of course, love to see more female developers. But you don't have to code to be part of building a software application. There are many ways to use a technical and analytical mind, including being part of the growing number of people who test those products. In the Internet of Things, quality is even more important than it ever has been – when software runs your car, your heating system, your television and your home healthcare appliances, it better work. Now is not the time to put artificial barriers in place that exclude people because they don't like certain subjects in middle school. And that's exactly the message we send when we harp about girls in STEM.

Back To Index

For almost 30 years, **Lorinda Brandon** has worked in various management roles in the high-tech industry, including customer service, quality assurance and engineering. She has worked at some of the leading companies in the industry, including Mashery, SmartBear, RR Donnelley, EMC, Kayak Software, Exit41 and Intuit, among others.

She specializes in rejuvenating product management, quality assurance and engineering teams by re-organizing and expanding staff and refining processes used within organizations.

Follow her on Twitter @lindybrandon.

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

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

Well, the wait is now over . Introducing….

"The Guiding Star"

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

E-mail your question on –

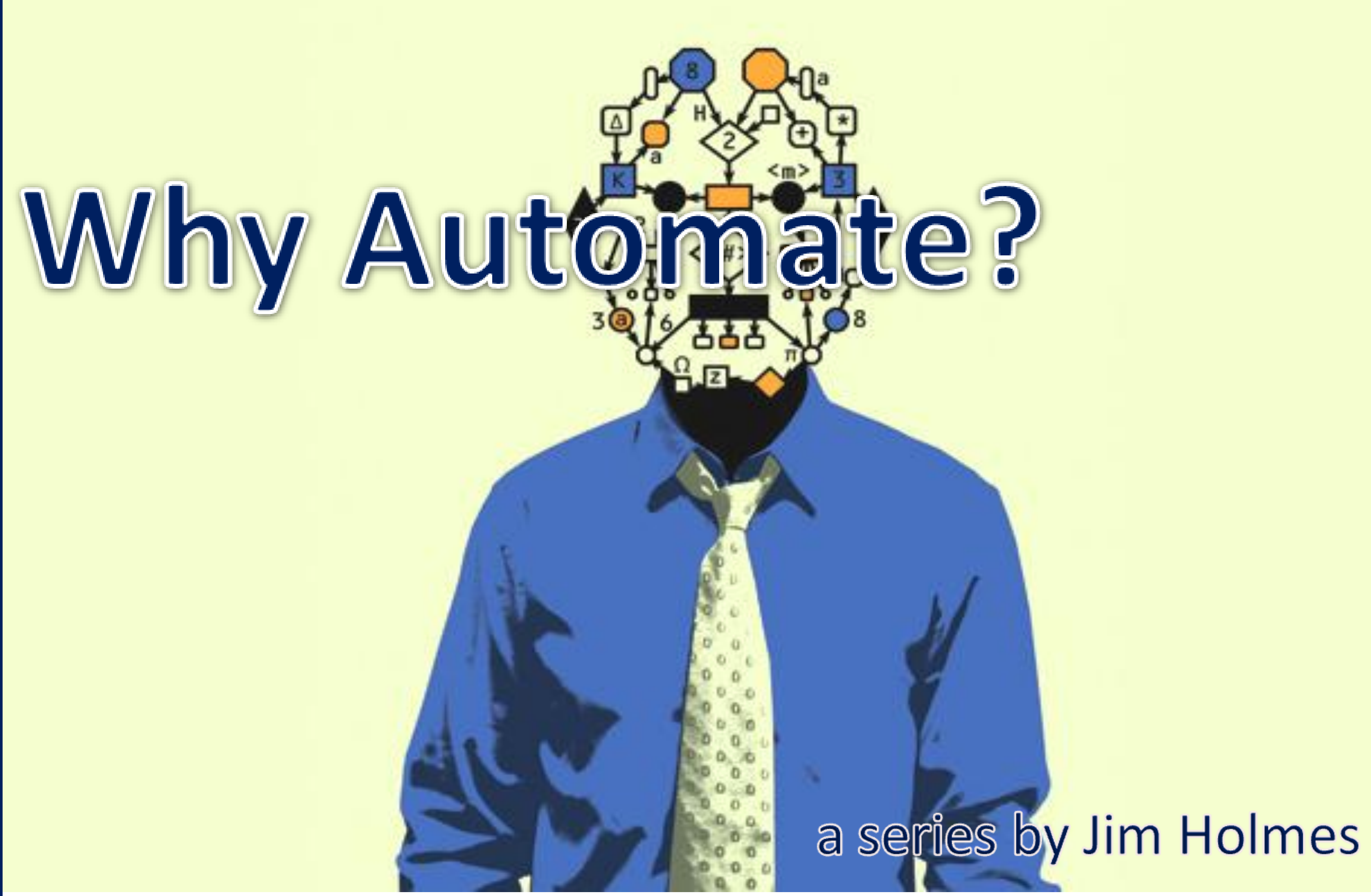
theguidingstar@teatimewithtesters.com

Please Note :

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

Image courtesy : MMVI New Line Production

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



UI Automation: Before You Even Start

Intro to Series

Welcome to the first article in a somewhat open-ended series on being successful with user interface (UI) automation. The goal of this series is to help you start asking the right questions to begin building high-value, maintainable automation in place for your projects. Once you're past the basics, I hope to help you find the right questions on how to evolve your testing over time to continue adding value to your project.

You'll notice I've mentioned "questions" a lot; answers, not so much. UI automation is a difficult domain to work in, and there aren't any magic "best practices" other than one: use your brain. This series will hopefully give you some ideas on how to look at your specific environment, and how to create a new (or tailor an existing!) automation strategy.

Why do UI Automation in the First Place?

Your first question about UI automation is one that's too often missed: why do UI automation at all? Aren't unit, integration, and manual/exploratory tests enough?

We in the software industry don't ask "Why?" anywhere near as often as we should. UI automation is a significant cost in your overall development cycle. It's important you understand if it's worth the cost before jumping in!

UI automation ensures you've got tripwires around your most critical business value features in your system. Hopefully you've got a solid amount of test automation at other levels, such as unit and integration. Unit and integration tests are critical to a system's overall automation approach, but by definition they don't cross all boundaries of a functional slice.

I often refer to a line my friend Adam Goucher uses when discussing why teams should do UI automation: "Show me the money!" he says, quipping a line from the film Jerry McQuire. Creating solid UI automation ensures you're guarded against regressions around business-critical features, like your customers adding items to a shopping cart and checking out—in other words, giving your company money. Other business-critical use cases might be ensuring your security or permissions system is correctly guarding critical data, or that the administrative screens fronting your order processing screens actually do cause orders to be passed off to your business partners' systems correctly.

Teams should ***not*** look to UI automation as a replacement for the 4,942 manual test scripts they have in place. That is a painful path that's guaranteed to lead to failure. Those manual scripts should be evaluated to understand what critical "Show me the money!"

Once you've gotten a good grasp on why you should be doing UI automation for your project, it's time to move on to setting yourself up for success. Getting a solid set of expectations from management and within the team is critical if you're going to move forward.

Setting Your Team up for Success

Setting Expectations with Management

Before you jump off the UI automation cliff, you need to ensure your management chain understands the costs associated with getting this sort of automation in place. You many also want to ensure they understand the benefits, too… (By "management" I mean your leadership, stakeholders, and customers.)

UI automation is a software development task, and it takes time. You're going to need time on the schedule to evaluate what tools to use. You'll need time on the schedule to get the tooling set up in your environment, you'll need time on the schedule to get your team's skills up to speed on the new tooling, and you'll need significant time (and patience!) on the schedule to adjust your development processes. Finally, you're going to need time on the schedule to get your UI automation work done.

You'll note my overly repetitious repeating of the phrase "on the schedule." Too many times management expects that UI automation effort won't impact the schedule. This is a recipe for frustration, missed deliverables, and ever-worsening team morale. There's no way you'll be successful in the long run if management doesn't acknowledge there will be, at times, significant impacts to the velocity you're delivering at.

You've got to balance this decrease in velocity with understanding the wins you're getting in other areas: time freed from repeating senseless manual regression tests, clearer information on your system's overall risk to business value, and an increase in confidence around your automation safety net.

Setting Expectations with the Team

No UI automation project succeeds without involvement from the entire team. I'm a firm believer that teams throwing software over a fence to some "automation team" isolated from the rest of the group are bound to fail, or at least see only mild successes with an extraordinary amount of pain.

Starting a UI automation effort means you'll need to get your entire team involved *throughout* the entire process. Developers and testers will need to work together to get the best automation in place. That means getting discussions in place early in the process—before a single line of code is written or updated—to talk about how automation will be handled for the particular set of work items being considered.

Developers and testers have to get the expectation in place, from the outset, that they'll need to be working closely together to create testable user interfaces. Small things like strategically placing static or pattern based ID values on certain elements; creating backing APIs for setting up and tearing down test data; building switches to turn on and off system configuration options like CAPTCHA or rich text editors; all these and other similar tasks have to be accomplished by the right people on a team pairing up.

The Hard Work's Before You Start

User Interface automation can bring great value to your projects, but you're doomed for an incredible amount of pain if you don't lay the groundwork for success before you write a single line of code.

Ensure you know *why* you're getting into UI automation. Ensure you know what you want to accomplish with it. Above all, take a great deal of care in setting reasonable expectations with your management and team.

As I said some time ago on Twitter, "UI automation is hard, regardless of what toolset is used. The core problem domain is hard. You WILL struggle, you WILL fail, you MUST learn" (https://twitter.com/ajimholmes/status/362535202834231296)

Jim Holmes is the Director of Engineering for Test Studio at Telerik. He has over 25 years in the IT field in positions including PC technician, WAN manager, customer relations manager, developer, and yes, tester. Jim has held jobs in the US Air Force, DOD sector, the software consulting domain, and commercial software product sectors. He's been a long-time advocate of test automation and has delivered software on a wide range of platforms.

He co-authored the book Windows Developer Power Tools and blogs frequently at http://FrazzledDad.com. Jim is also the President of the Board of Directors for the CodeMash conference held in the middle of winter at an indoor waterpark in Sandusky, Ohio.

Back To Index

Introducing Change in your Testing Department

a series by Bernice Ruhland

Session-Based Testing, Testing Frameworks & Tools

Many testers are champions for change in their organization to help testers adopt different testing approaches or tools. You may have attended the Rapid Testing Intensive course or read articles about session-based testing. After learning new methods, we are energized to start using them. Sometimes we receive resistance from other testers because they are not as quick to adopt taking a cautious approach. This series will discuss how you can use session-based testing and tools like mind maps while providing suggestions on how to implement change within your testing team or department. This series is not intended to provide training on session-based testing. I would recommend that you attend a Rapid Testing Intensive or Rapid Software Testing workshop. This article will focus on basic terminology and definitions that will be used throughout this series. At the end of this article is reference material for additional information and background regarding session-based testing and mind maps.

Session-Based Testing

James and Jonathan Bach developed session-based testing in 2000 with the intention of placing structure and accountability around exploratory testing in a manner that allows creativity while providing a rough countable unit of testing (uninterrupted time). This structure should not be confused with test plans, test cases, and traditional metrics that focus on number of test cases that passed testing.

Product Coverage Outline (PCO)

A product coverage outline (PCO) provides an outline of any aspect of the product that we may consider testing. The PCO describes the "surfaces" of the product, meaning the bits that we can get at, or otherwise cover for the purposes of testing. It does not document any of the tests. A tester can use this outline to help identify tests and to ensure all areas have been tested for a particular feature (i.e. a data entry screen). The PCO is a living document because information should be added or removed as the product evolves. (A PCO is part of the Rapid Software Testing Methodology.)

Risk Analysis

The PCO can assist in identifying the kind of bugs we are concerned might be in the product we are testing. These potential bugs can be summarized into risk areas, which is a cluster of similar risks (i.e. performance). For each risk area, determine the likelihood and impact to determine how much time to spend testing.

Testing Charters

Testing charters provide a goal for a test session. It helps keep a tester focused on a specific testing mission. A charter may take more than one test session to complete. If a charter takes more than three or four sessions, it might be best to reduce the testing scope. Anything a tester needs to do in order to test can be described in a brief charter for a session. Over time, you will probably find that your sessions tend to fall into certain patterns. James and Jonathan identified these major types of test sessions: Intake, Setup, Survey, Analysis, Deep Coverage, and Closure. The point of the different types is that it helps test planning to be able to look at something and say "I think that is going to be at least one intake session, a setup session, two survey sessions, an analysis session, seven or eight deep coverage sessions, and a few closing sessions."

Below is a brief description of each session type based upon my understanding from attending the Rapid Testing Intensive workshop and through my professional reading.

- Intake Session

When assigned to test a new feature, there is typically a lot to understand regarding the mission and work scope. An intake session charter supports this learning activity. An example of an intake session charter: Prepare to test the new data entry screen. This charter would require you to talk to the right people to understand more about the feature that is being tested.

- Setup Session

The test environment must be set up before testing can begin. A setup session charter supports these activities. For example a setup session charter could include the following activities: identify or create test data; identify any test tools; and identify other activities needed to prepare for testing. These activities need to be performed to support finding bugs; but you are not testing for bugs with the setup session charter.

- Survey Session

The purpose of a survey session is to learn the product. Give yourself time to learn before you start to produce any artifacts for deep coverage testing.

- Analysis Session

The analysis session charter helps the tester get his/her ideas in order for deeper testing. At this stage you will create or review a PCO. Any documentation created can change as we learn more through testing.

- Deep Coverage Session

A deep coverage session purposely tests at a deeper level. It is systematic coverage with the objective of finding obscure bugs; whereas the object of a survey session is to learn what you need to learn in order to do reliably deep testing. The tests are more complex in nature and can include combinatorial testing and state-based models.

- Closure Session

A closure session provides time to perform wrap-up activities such as: regression testing, lower-priority testing, follow-up on any outstanding strategy items, and complete any documentation.

Test Sessions

A test session is an uninterrupted period of time for a tester to complete a testing charter. Test sessions typically last about90- to 120-minutes. A tester may complete 3 sessions a day, while allowing time to answer email, attend meetings, and breaks. However, depending upon the product's complexity, longer sessions may be appropriate. Paul Holland's interview with LogiGear Corporation from CAST2012 provides an example on why it could be appropriate to perform two sessions a day. Paul discusses why they selected to perform one session in the morning and a second session in the afternoon. In this short interview he discusses how he implemented rapid software testing at a prior company.

Session Report

A session report documents the testing outcome of a test session. This can include a brief overview of what was tested, what was not tested, problems encountered, and suggestions for additional test charters. Conciseness is important while providing meaningful information to understand test coverage and limitations. Do not confuse a session report with a traditional test report that may be a small book.

Debriefing Session

A debriefing session is a short meeting (about 15 – 20 minutes) between the tester and manager to review the testing results. In Jonathan Bach's article "Session-Based Test Management" he uses the acronym PROOF to describe the structure of the meeting.

☐ ***P****ast*. What happened during the session?

☐ ***R****esults*. What was achieved during the session?

☐ ***O****bstacles*. What got in the way of good testing?

☐ ***O****utlook*. What still needs to be done?

☐ ***F****eelings*. How does the tester feel about all this?

Session Metrics

Session metrics provide data about the exploratory testing status. Information such as the number of sessions completed, number of problems found, and percentage of time spent investigating problems can be captured. Each piece of information that is track has a specific purpose in providing the testing status.

Other Terminology

The below terminology is not necessarily part of session-based testing but is terminology that will be discussed in future articles.

Timeboxing

Timeboxing is a way to allocate time to perform testing activities or other administrative tasks. Ideally the timebox is an uninterrupted block of time to perform an activity. It does require discipline and planning to complete the activity within the allotted time. As an example, you might identify a timebox of 1-hour to initially learn a new feature before working with a Business Analyst on questions. Without a timebox you might find yourself spending more time than is warranted on the task.

Change Agent

A change agent is someone who uses his/her influence to encourage other employees to adopt change. Basically s/he is a champion for the change being implemented such as session-based testing. Their role can include ensuring proper communication throughout the process, identifying and removing barriers to change, and overseeing the change.

MindMaps

A mind map is a visual drawing showing relationships by using either a software tool or paper. From a testing perspective, a mind map provides a visual picture to represent areas such as test strategy, test ideas, or allocation of testing assignments. It is a powerful tool to organize testing ideas by capturing them as a test team generates them. Those testing ideas can then be organized into topics or assignments. There are many free and paid software tools that allow you to create an electronic mind map that can evolve based upon what you are learning during testing.

References for this article series:

- Paul Holland's interview with LogiGear Corporation from CAST2012. Go to YouTube and search for "Paul Holland on Rapid Software Testing".
- I attended Rapid Testing Intensive 1 & 2 as a remote tester and learner with James Bach and Jonathan Bach. Documentation reviewed for the training: Rapid Software Testing V3.0 slide presentation by James Bach and Michael Bolton. Rapid Software Testing Appendices by James Bach.
- James Bach website: www.satisfice.com
- Session-Based Test Management article by Jonathan Bach http://www.satisfice.com/articles/sbtm.pdf
- An Exploratory Tester's Notebook by Michael Bolton
- http://www.developsense.com/presentations/etnotebook.pdf
- Lean Test Case Design by Darren McMillan (Tea Time with Testers: October 2011)
- Mind Mapping 101 Part 1 by Darren McMillan (Tea Time with Testers: November 2011)
- Mind Mapping 101 Part 2 by Darren McMillan (Tea Time with Testers: December 2011)

Back To Index

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

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

Bernice can be reached at:

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

Imparting Knowledge QUALITY LEARNING

Latest from Quality Learning

Selenium WebDriver Online Training

Special offer

Buy two seats and get 100% off on third!

Note - This offer valid till 10th Nov 2013 only

Can we Ship?

by Sashidhar Penumala

1. Introduction

Software is all over the place and has become a foremost wide-reaching engineering practice.

Few decades ago we hardly hear people exchanging email addresses, people never worried about upgrading licenses or versions. But now we see software in use for vending food items in homes and hotels, it is use in transport, communication, healthcare, education, insurance, and banking and even in terrorism.

2. Need for Testing

With Software intruding much into human lives it has become extremely important that software meets expected behaviour and does not result in any ambiguous behaviour. It's important that transport system exactly checks and bills the customer as per the usage, it is important that coffee vending machine serves only coffee when the opts for it, it's important that online money transactions are successful and reach the destined person safely and timely, it's important that healthcare systems are so reliable during surgeries, it's important that fuelling systems are so precise that they do not incur loss when 0.0001 units is extra served for each serving. In a nutshell it's equally important that every software meet expected standards and does not result in any ambiguous behaviour which can lead loss of business or loss of reputation based software. This ambiguity makes difficult to determine or predict the software release. So to take a decision whether software is predictable to release i.e. to ship; we need testing. There were many instances where there was a *software Crisis* where there are problems in software development that caused the entire system to be late, over budget, not responsive to the user and/or customer requirements, and difficult to use, maintain and enhance.

3. When we can stop testing?

The question "when can testing be stopped" so the product can be released is gaining its presence now. Software testing is intended at assess quality which in turn has many attributes.

The possible input combinations are too high for testers to apply them all. Complete testing is, however, not realistic. The number of tests required achieving any given level of test coverage increases exponentially with software size so it is practically impossible that we can do complete testing. Thus we can say complete testing is not possible.

Testing may be effective in showing the presence of defects, but it is inadequate for showing their absence

In general we can know if testing done has been sufficient or we need to do more testing based on the testing done on the software. It's an accepted fact that defects are measure of quality. The less the software more is the quality. But how does one know whether we uncovered all the defects. We will be using few bug prediction methodologies and decision making techniques:

Prediction Techniques	**Decision Making Techniques**
Density	*Zero failure method*
Pooling	stopping rule method
Seeding	Cost vs. Quality Method

In this article I will discuss about only Defect prediction techniques as below:

3.1. Defect Density

One of the easiest ways to judge whether a program is ready to release is to measure its defect density i.e. the number of defects per line of code. Suppose that the first version of product, Software 1.0, consisted of 100,000 lines of code, you detected 700 defects in version 1.0. Then defect density is measured as 7 defects per thousand lines of code (KLOC).

Suppose that Software 2.0 consisted of 50,000 additional lines of code, that you detected 475defects. Then total defect density of that release would be 475 total defects divided by 50,000 new lines of code. So we can say defect density is 9.5 defects per KLOC for version 2.0

Now suppose that you're trying to decide whether Software 3.0 is reliable enough to ship. It consists of 100,000 new lines of code, and you've detected 600 defects so far, or 6 defects per KLOC. Unless you have a good reason to think that your development process has improved with this project, your experience should lead you to expect between 7 and 10 defects per KLOC. So we can expect that we can get 100 more defects such that defect density will be in the range 7 to 10. In this case we can say system is prone to more bugs and testing should continue.

The more historical project data you have, the more confident you can be in your pre-release defect density targets. So we should collect data as part of our metrics management

Back To Index

3.2. Defect Pooling

Another simple defect prediction technique is to separate defect reports into two pools. Call them Pool A and Pool B. We then track the defects in these two pools separately. It doesn't really matter how you make the division as long as both reporting pools operate independently and both test the full scope of the product.

Once we create a distinction between the two pools, we track the number of defects reported in Pool A, the number in Pool B, and the number of defects that are common in both Pool A and Pool B. The number of total defects can then be approximated using below formula:

Defects Predicted = (DefectsA * DefectsB) / Unique Defects

If the Software 3.0 project has 400 defects in Pool A, 350 defects in Pool B, and 150 common defects. The approximate number of total defects that would be present is (400 * 350) / 150 = 933.

We already analysed that total defects identified so far is 400+350-150=600. 150 are reduced as they are common defects. This technique suggests that there are approximately 333 (933-600) defects yet to be detected.

3.3. Defect Seeding

Defect seeding is a practice in which defects are intentionally inserted into a program by one group for detection by another group. The ratio of the number of seeded defects detected to the total number of defects seeded provides a rough idea of the total number of unseeded defects that have been detected.

Suppose on Software 3.0 that we intentionally seeded the program with 50 errors. Suppose that at a point in the project when you believe testing to be almost complete you look at the seeded defect report. You find that 31 seeded defects and 600 normal defects have been reported. You can estimate the total number of defects with the formula:

Defects Predicted = (Seeded Defects Injected / SeededDefects Found) * DefectsFound

This technique suggests that Software 3.0 has approximately (50 / 31)* 600 = 967 total defects.

A common problem with defect seeding programs is forgetting to remove the seeded defects. Another common problem is that removing the seeded defects introduces new errors.

By Analyzing above methods we can know how many bugs might be present in a build by which we can take a decision on whether sufficient testing is performed or not by which release decision is understood.

Sashidhar Penumala has 6 years of experience in Software Quality Assurance in several testing roles.

Currently he is a Technical Lead at ValueLabs. He is involved in strategizing testing and delivery of testing projects. He believes that excellence can be achieved by the eagerness to learn new things and the attitude shown in implementing them at work. He is fond of Exploratory testing which is of his core skill, he manages mobile testing projects, security testing projects"

He loves training and sharing knowledge and He can be emailed at sashidhar.value@gmail.com and tweets as @mycupofLife

'The Importance of Testing in the App market: does quality really matter when launching the next hot app?'

By Martin Wrigley, Executive Director, App Quality Alliance

The crucial importance of professional software testing practices in the hyper-fast and competitive mobile app market. How can it give developers an edge?

A recent mobile app survey found that a faulty app would lead a staggering 96% of users to write up a bad review and that 44% of people would immediately delete the app. The results prove that consumers will not tolerate low quality risky apps that could potentially impact their phone's normal activity.

What does this mean for the developers who are putting these apps out there? Surely they stand to lose not only their source of income, but also their reputation? It therefore begs the question of why app developers don't spend a little more time on the professional software testing of their app before it's shipped to the general public. In addition to the risk of alienating their first generation of users, developers also face a huge amount of competition from other app developers and won't get a second chance to make that all-important first impression; should their app be faulty.

Be it a big brand, or the guy developing apps in his garage, it's no longer good enough to maintain development processes that refuse to acknowledge a strong testing element. Without legitimate testing processes developers can't gain an edge over their competition. And as far as brands are concerned, a lack of testing and production of a poor quality app, can even damage their brand reputation.

We are often told that the customer is king and Google Play's customer policy is no different. Customers can request a full refund within15 minutes of downloading an app, with no questions asked. With 700,000 other apps in Google Play, can a developer really afford to miss his one chance of gaining customer loyalty? With all these barriers in place, the case for a thorough testing process to ensure you don't fall at the first hurdle starts to become common sense.

QA practices - how can you ensure the right level of testing at the same time as having speed to market?

From my experience, one of the most crucial aspects for a developer is speed to market. But how do you ensure you balance this with the right level of testing? Basically in exactly the same way as any other manufacturing process. Rigorous testing and inspection will in general not only produce a higher quality product, but interestingly bring it quicker to market.

With apps as with any other product, speed to market is pointless if the product doesn't work! The earlier you test, the earlier you find faults, and the earlier you fix the problem and ship to market.

The embryonic stage of an app is really the crucial time to engage with the testing process. Building and building in a standard testing procedure at the start saves of hours of work that needs to be done later if the app is faulty. Also with the increasing complexity of apps, the case for testing becomes more apparent.

Simple steps a developer can take to ensure increased usability of their app

Many developers, if not most, are great at development and producing a superb set of functionality, but aren't experts at testing – and why should they be?

In October 2012 at the Appsworld event, we launched a new initiative: a Quality App Directory. It isn't a shop and it won't cost the developer to list their app; but to be included in the Directory the app must have gone through and passed a good QA process. Now what, you may ask, is good? And who can define that? And how can you use that to help improve your own app?

Using AQuA's best practice guidelines and the testing criteria gives an app developer a pre-defined set of tests that complement the functional testing that every good developer does as they go along. In essence it defines the non-functional tests that many developers might not think to carry out, but things that real users do. There are some clear common errors that many people miss, purely through not thinking to test for them.

Add to this the increasing variety of platforms that exist, from iOS, Android and Windows, through to Firefox, Blackberry and Tizen. If a developer wants to ensure increased usability of their app, they not only have to incorporate a testing process that introduces quality from the outset, but must ensure that the app is wholly consistent with the platform. A successful port to a new platform should involve adapting to the platform's way of working with the UI, as well as the purely technical interfaces. This involves incurring platform specific testing and being consistent with the platform and the app will automatically have an increased level of usability.

So as you can see, good QA can really help you avoid some common pitfalls. Below I've listed out what I feel are the most common failures developers come across when developing apps and tips on how to avoid them.

1. User interface inconsistency;
   make sure menu options, button labels, 'soft keys', menus etc are consistent and clear.
2. Lack of clarity of graphics and text;
   make sure that all the text is readable, clear and not cut off by the edged of the screen or overlapping other screen items
3. App browsing confusion;
   although the navigation through the app is obvious if you've been working on it for weeks or months, not everyone else may find it so clear.
4. Language inconsistency and spelling errors;
   if you support multiple languages, make sure that it is consistent and you don't have the odd label in English hidden away… And use a spell checker!
5. Privacy policy omission;
   you must always have a privacy policy in the app, this is getting a real hot topic now.
6. Hidden features;
   doing stuff behind the scenes without letting the user know will never win you any favours, even if your intentions are good.
7. App crashing;
   you would be surprised how may apps can be made to crash when even some simple things happen on the device, memory cards, attachments, keyboards are common causes.
8. Help is not there;
   whilst it is obvious to some, other people like to read help information and so providing help is a must.
9. Network connection: lack of notification;
   again, so many people don't test the phone dropping out of coverage. If you miss it and the app dies when the connection drops, the user ends up re-booting their device. With new networks and more handovers between technologies this is going to be a new hot topic.
10. Screen orientation distortion;
    surely everyone check this one? No sadly not. Distorted images when changing from portrait to landscape and vice-versa still manages to hit our top ten simple errors that let apps down.

How important is testing to the likes of Apple and Google when it comes to the App store and Google Play?

When it comes to the App store and Google Play they take diametrically opposed attitudes, but in both cases it's the developers who benefit from a good QA level. Some developers regard Apple's content policies and their technical requirements as unclear and there can be some significant delay if the developers haven't tested their app prior to submission. There are certainly no second chances with Apple who run a curated app store with a basic set of functionality tests that may not pick up on every fault and flaw in the app. Google may have a more open door policy enabling you to enter with no testing, but if your app doesn't work, users will get their refund and post a bad comment. If you are relying on advertising revenue, it is even more important that users carry on using the app, so good QA is a key factor in generating revenue.

In both cases getting the level of testing right is key, and this is where AQuA's Testing Criteria come into play. For Apple, it's crucial that developers pass its entry functionality test and with the help of a set of testing criteria the risks of not gaining entry are greatly mitigated. As for Google, a fault-free app means not being discarded by millions of customers in favour of the next app in the list.

What lessons can app developers and testers learn from years of testing in the software and hardware space?

Each day sees more and more developers take their chances in an already overly populated app market. With many brilliant ideas for apps available, one thing that sets the best developers apart is their ability to produce a true quality app. An app that fits the customer need and works well in its environment.

The scope and scale of the mobile app market and the speed of comment and criticism of a poor app, means that no developer can afford to ignore quality. Testing is the route to achieving this, and as the environment continues to change with different days bringing different devices, so too must the developer's attitude to testing.

What developers can really draw on is the experience of testers in the areas of software and hardware. Their experience really does count and developers can stand tall on the shoulders of those testers who have 'been there and done that' before. By using the experience of others, you may not necessarily end up being the hero, but you will definitely get the best job done.

Back To Index

Taking
a break?

a click here
will take you there

"Do what you love and Love what you do" is what Ashok lives by. A true Arien, he is passionate in whatever he does and tickled by challenges. Solving technical problems via code was the challenge he enjoyed in the first decade then discovered guaranteeing the solution to the problem was more challenging as it tended to be fuzzy. He founded STAG Software in 2000 a pure play test boutique and has been focused on developing HBT a personal scientific methodology.

"Simple is complex', he revels in taming complexity and enjoys the learning and discovery it offers. A strong believer in opposites, the Yin and Yang, he strives to marry the western system of scientific thinking with the eastern system of belief and mindfulness.

An avid cyclist, he is into long distance endurance cycling. A double Super Randonneur this year, with the longest ride of 1000km ride in 67 hours riding three consecutive, he enjoyed hallucinating in the last night, which he says it was Zen. He is eagerly looking forward to participating in the PBP 2015 at France. Now the running bug has bit him and he is practicing to do his maiden half-marathon the end of this year.

Know more about Ashok and his views on software testing in this exclusive interview with Tea-time with Testers…

Over a Cup of Tea with T Ashok

Image credit – Bug.deBug Conference

Please tell our readers about your journey as a Tester.

Way back , the first tester who tested my product (a DOS based software) filed a bug that said ‘In response to the message displayed “Press any key to continue”, I pressed CAPS LOCK and only the light came ON’. This is when I developed a severe dislike for testers. As a developer penchant on writing neat and clean code, I complained to manager about the poor tests that “ the QA did”(remember I am the dev guy). My manager said “Do not complain, come up with a solution”. That was when I setup the test engineering group (guess we were the first in Bangalore) and became a keen student exploring ways to scientifically test software. This group became very respected and well liked by the wonderful developers we had.

Please tell us about STAG Software. What made you start a Software Testing firm?

When I was having a wonderful time building the test engineering team, the company I worked was acquired by a much larger company.

This is when I decided to “do what I like and love what I do” and founded STAG.

The vision of STAG to make the world of software “clean”. Our focus is on inventing methods & tools to scientifically detect/prevent defects n software.

What do you think would be the skills that would make testers valuable?

Testing is an interesting profession that requires you to read carefully yet be a mind reader, think logically yet be random, have a healthy dose of suspicion so that people can trust the system, criticize the developers but emphathise with the end user, be disciplined and creative, be fearless but worry about delivering real value to customers.

Please tell us about HBT.

HBT or Hypothesis Based Testing is a personal scientific methodology that is unique in its approach to ensuring cleanliness of software It is not a process system; rather it is a methodology for an individual to deliver clean software. It is based on sound engineering principles geared to deliver the promise of clean software. The central theme is of constructing a hypothesis of potential defects that may be probable and then scientifically proving/disproving them.

My tryst with a scientific approach to uncovering defects commenced in 1997. When I founded STAG, I focused on the principles & techniques that today form the core of the HBT and then packaged them into specific disciplines of thinking. Then came the methodology to enable a structured application of these disciplines in a staged manner. Now I am working on the HBT Version 3 that simplifies application greatly via standardization. In the last 13 years, it has evolved FIVE times.

You are known as a prolific writer and excellent conference speaker in community. What makes you master those skills? Any advice to aspiring/novice writers and conference speakers?

“Enjoy what you do, Do what you enjoy”. Practice is then not a chore. With practice, you acquire skills and believe in yourself. And this is the key to speaking from your heart, with your mind.

Agile and Automation are generally considered as solution to any testing/business problem now a days. Your Â opinion? How to find balance between automation and manual testing?

Agility is about doing smart work. Automation is about doing the hard work. The need of the hour is stay focused, be sharp and do more but no more. Agility is about being responsive, this requires us to lean and not do more i.e. focus what is necessary and drop the rest.

I am sorry we have terrible phrases that have crept into our discipline - manual testing and automated testing being two of them!
We should not be using the word "testing", rather it should be "test execution". Using technology to do the grunt work is smart. Manual really implies "intellectual" and it is expected this is not be compromised.

What is your opinion about test metrics?

Metrics are like mirror. Mirror enables us to reflect what we do and therefore be better. To correct we need a good mirror i.e. we need to collect metrics that reflect reality well and facilitate behaviour change. Like the mirror in Cinderalla story, it can be abused too.

In industry, testers are generally evaluated based on number of defects found/missed, count of test cases written etc. What is the sensible and best approach towards evaluation of any tester in your opinion?

Let me set the context that we need to evaluate the process & outcomes and not just individuals. Good process/outcomes result in clean software in the given constraints of time/effort.

Is the person curious? Does (s)he question or assume? Is (s)he organized and logical? Can (s)he justify as why they do did that ? Are there well versed with techniques, technology and tools to do the job? Do they argue or succumb? It is necessary to justify goodness of design and demonstrate confidence in the software via good quality defects.

If not a tester, what would you have loved to see yourself as?

It is *not a OR * question. I see dev/test as two sides of the software engineer coin. All I strive to is be a good (software) engineer.

Would you like to tell us about your love for cycling?

Two years I got into cycling and discovered I liked long distance endurance cycling. Discovered that it is a form of meditation that helps you discover yourself.

I do brevets i.e. self supported long distance rides and my longest distance has been 1000km in 67 hours. I became a Double Super Randonneur(SR) this year. (To become a SR you need to complete 200, 300, 400, 600 km brevets in the designated time period in the 'cycling year')

Things that would make one excellent test lead or manager?

- Empathy - to understand your people and to think like a customer.
- No-fear, can-do attitude.
- Goal focused, no excuses, get-it-done.
- Think with mind, work via heart.
- Exude confidence.
- Be honest.

What is your opinion on importance of hands on testing for leads and managers?

Two kinds of people have gotten the best work out of me.

- Those with great hands-on knowledge who understood my technical issues and helped me fix it. And I got better.

- The second kind, who were excellent people person, they motivated me to give my best for them. They were not hands-on people.

Leads/managers have to gain respect of the team, and you can gain via technical route by being hands-on or being a great people and management person who provide the motivation and facilitation.

You have been writing in 'T Talks' column of TTwT from almost 3 years. And it has been appreciated by a lot many readers. How do you feel about it? How do you feel about being member of Tea time with Testers family?

I have thoroughly enjoyed being the member of TTwT and delighted it has been well received and anticipated every month by the testing community. Kudos to you Lalit.

I look forward to end of every month to write a new article and each one has been an original article for TTwT.

Your message to our readers would be?

“Enjoy what you do, Do what you enjoy”. Have fun. Be in the present. Be open, enjoy the contradictions.

Do more work by doing less. Think better. It is not just how good we do the work, it is the value that we deliver that matters.

Read and Enjoy the “Tea time” with “Testers”.

Call for Articles !

Have you got something to say?

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

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

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

Submit your articles, stories, thoughts around software testing.

now its your chance to be heard...!

Click HERE to read our Article Submission FAQs !

sciencephotogallery

Lead and ye shall grow

Quite frequently in conversations with test professionals I come across questions that relate to career growth - "How do I grow in testing? What should I do to grow? What areas in testing should I pursue to grow quickly? ...". These are interesting conversations as there is no specific answer.

Let us dig a little deeper to understand what career growth may mean. Is growth about earning more money? About better designation? Or more 'power' in terms of larger project/team size…? I bet your answer will be "All of these". Hmmm, these are outcomes of growth, but the question still remains - "what is growth?". Is growth deeper knowledge? Or skills/abilities? Or is it the confidence to get anything done?

I would like to believe that the last one is most appropriate. It is about having the confidence of getting anything done. If you can handle stuff that is more fuzzy, difficult, constrained, large, with huge expectations, then you grow faster and higher.

It is not just doing work, it is about getting work done. It is about being a leader.

Reflect. A young baby needs support to get things done, as he grows, he manages to be self-sufficient. As he becomes a young adult, he is able to manage others and later as an adult when he establishes his family, he leads, not just manage. As you grow this is what happens : Managing yourself --> Managing others(or things) --> Leading (others/things). The ability to "do", "manage doers" to "leading doers" is what defines growth. Initially it is about managing, later evolving into leadership.

Leadership is about influencing others. It is about believing in yourself and not really care about what others think. A far cry from being dependent on what others think about you to figure out if you are good. It is about having belief and confidence in yourself. It is not about 'toeing' the line, it is about leading. It is not just agreeing to other's view, it is healthy arguments to put forth yours views/thoughts.

Bodily growth cannot happen when you just eat more, it happens only when the food is digested and assimilated. Likewise, it is not about just knowing more, it is about deep understanding aided by reflection that allows us to assimilate. This results in more than just knowing, it is about forming heuristics as when to/not-to apply in way similar to purging unwanted stuff.
This results in one become more confident in the application of the knowledge.

Growth is evolution and evolution is "Add/Modify/Delete". Visible external growth is "Add", the accumulation of mere knowledge. Real 'internal' growth is "Modify/Delete", the assimilation that changes the internals, and purgation, the "Delete" that discards old views/ideas, strengthening you from inside. Growth is not merely external, it is building inner strength.

With inner strength comes confidence and the power to influence. To influence the project team we work with. To influence the product we are building. To influence the company where we work. To influence the customer who uses our product. To influence the test & software engineering community.
The expanding spheres of influence :
Individual "I" --> Project team "US" --> Product "OURS" --> Company "US & OURS" --> Customer "THEM" --> Community "WE".

Now you do not care about what others think. You have grown. Stuff happens. You do not 'just do actions and get things done'. You deliver 'business value'. You feel good. And possibly get noticed. Anyways you enjoy what you have done. And the wonderful outcomes of growth happen.

As another year comes to a close shortly, it is natural to think how we have grown and chart the path for the future. It is not just merely knowing new technology, techniques, domains, tools, it is about how to use the power of knowledge to influence all around you. This year, think not on the outcomes of growth, chart out the plan to influence.

"Lead and ye shall grow".

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

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

He can be reached at **ash@stagsoftware.com**

Back To Index

by

QUALITY TESTING

Quality is delighting customers

Testing PUZZLES

by Sebi

Claim your Smart Tester of The Month Award. Send us your answer for Puzzle b4 20th November 2013 & grab your Title.

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

Try to get /etc/passwd or C://Windows/win.ini from this website officeathand.att.com. If you can't, the best answer will be considered the one with the best tries and explanations of tries.

And winner for last month's puzzle is:

Sharaniya Srinivasan

Heartiest Congratulations!!!

Back To Index

Biography

Blindu Eusebiu (a.k.a. Sebi) is a tester for more than 5 years.

He considers himself a context-driven follower and he is a fan of exploratory testing.

He tweets as @testalways.

You can find some interactive testing puzzles on his website www.testalways.com

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

in nexxt issue

articles by -

Jerry Weinberg

T Ashok

Jim Holmes

Bernice Ruhland

Mike Lyles

JeanAnn Harrison

Guy Mason

our family

Founder & Editor:

Lalitkumar Bhamare (Mumbai, India)

Pratikkumar Patel (Mumbai, India)

Lalitkumar

Pratikkumar

Contribution and Guidance:

Jerry Weinberg (U.S.A.)

T Ashok (India)

Joel Montvelisky (Israel)

Jerry

T Ashok

Joel

Editorial | Magazine Design | Logo Design | Web Design:

Lalitkumar Bhamare

Core Team:

Dr.Meeta Prakash (Bangalore, India)

Dr. Meeta Prakash

Testing Puzzle & Online Collaboration:

Eusebiu Blindu (Brno , Czech Republic)

Shweta Daiv (Mumbai, India)

Eusebiu

Shweta

Tech -Team:

Chris Philip (Mumbai, India)

Romil Gupta (Pune, India)

Kiran kumar (Mumbai, India)

Kiran Kumar

Chris

Romil

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

To get **FREE** copy ,

Subscribe to our group at

Google

Join our community on

facebook

Follow us on

www.teatimewithtesters.com