

Tea-time with Testers

Jerry Weinberg

The Golden Key

Fiona Charles

Six Impossible Things Before Breakfast

Lisa Crispin

Learning to Communicate Better with Programmers

Claire Moss

Esprit de Corps: From Adversary to Ally

Anurag Khode

Mobile Apps Testing-Need of Time

David Vydra

Increasing Test Effectiveness with Affordable Custom Tools !

Adam Yuret

Does ATDD=Waterfall ?

T Ashok

Landscaping: A Technique to aid Understanding

Joel Montvelisky

Testers !! Know Your Business !



EACH ISSUE IS
A **GREEN** ISSUE

JUNE 2011 | Year 1 Issue V | www.teatimewithtesters.com



Created and Published by

Tea-time with Testers.

Hiranandani, Powai- Mumbai -400076
Maharashtra, India.

Editorial and Advertising Enquiries:

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

© Copyright 2011. Tea-time with Testers.

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

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

Dear Readers,

They say that, "If you do something wholeheartedly & with honest desire, the universe opens its all doors for you."

How True! Having started in Feb 2011 and four issues published so far, Tea-time with Testers has now become one of the *favorite magazines of Global Testing Community*.

Recently, Suzan from Australia wrote me that Mr. James Bach recommended Tea-time with Testers at his *Rapid Software Testing Course* (see page 21). I feel overwhelmed when readers write to me saying that their colleague, friend or mentor recommended Tea-time with Testers. Some call it as a *treat for Testers* while some others as *most wanted thing they were looking for from ages*. It won't be an exaggeration if I say that "Tea-time with Testers" is the *only magazine which is being discussed in community* for its quality, feel, content, authors and many other original ideas, from its very first launch.

Once again, I would like to thank all of them who liked, enjoyed and have recommended Tea-time with Testers. I take it as a receipt of the hard work that our team is doing.

Well, before you scroll down, let me introduce our new team member Mr. Anurag Khode, who happens to be a passionate tester like us and also a well-known brain behind www.mobileappstesting.com & www.mobileqazone.com. Considering the future, scope and challenges in the field of Mobile Apps Testing, Anurag will be guiding you all through his expertise around said area.

I must not forget to update you regarding "*Teach-Testing*" campaign that we have started. We are getting some suggestions, opinions, guidance across the globe and we are committed to consider them. However, I appeal you to spend few minutes on Poll that we are conducting and cast your vote. I look forward to make this campaign a huge success by support and guidance from you all.

As usual we have given our best to offer you the best articles in this issue too. I appreciate the efforts that our writers have taken by devoting their valuable time and thank them all.

Well, that is all for now. Let's meet again in our next issue.

Feel free to drop me a note for any discussion around Tea-time with Testers.

Enjoy Reading! And have nice Tea-time!

Yours Sincerely,

Lalitkumar Bhamare



QuickLook

The logo for QAI, featuring the letters 'QAI' in a large, serif font. To the left of the letters is a vertical bar with a gradient from orange to white.

Editorial

What's making News?

Tea & Testing with Jerry Weinberg

Speaking Tester's Mind

Six Impossible Things Before Breakfast -14

Esprit de corps: From Adversary to Ally- 18

Does ATDD= Waterfall? - 22

In the School of Testing

Learning to Communicate Better with Programmers -28

Increasing Test Effectiveness with affordable Custom Tools -31

Mobile Apps Testing:- Need, Challenges and Opportunities - 34

Testing Intelligence: Testers !! Know Your Business! - 36

T' Talks

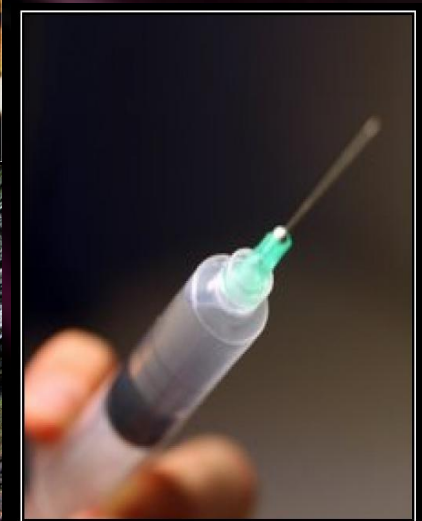
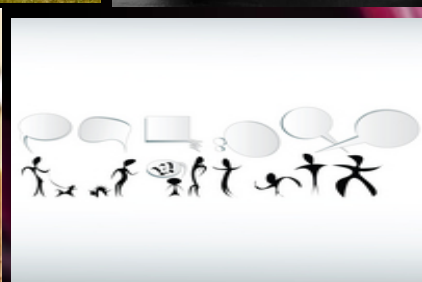
Landscaping: A Technique to aid Understanding – 40

Tool Watch

Testing Puzzle – S.T.O.M. Contest

Our Testimonials

Family de Tea-time with Testers



EFFECTIVENESS



Testing Puzzles
by Sebi



image: www.bigfoto.com

QAI

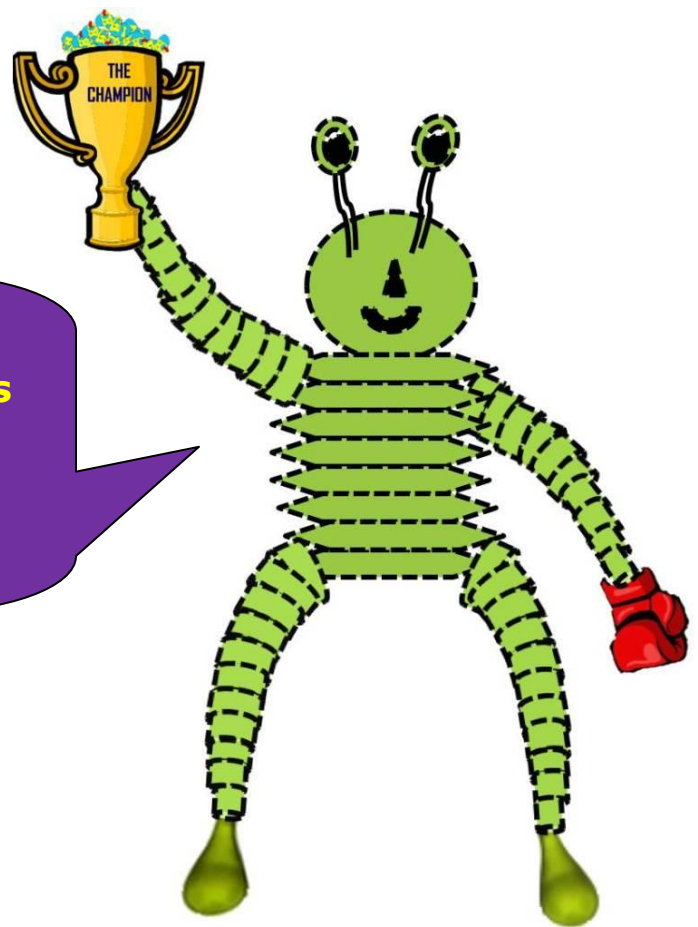
June 2011 | 4

QAI is a leading global consulting and workforce development organization addressing “Operational Excellence” in knowledge intensive service organizations. They work with majority of the top IT/ BPO companies, including about 2/3rd of the top IT/ ITES-BPO companies in India, by providing benchmarking, certifications, assessments and other Operational Excellence enabling services in the areas of CMMI, PM, Six Sigma, COPC and Performance Improvement.

Credits - Emeka Aginam for Vanguard

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

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



[Back To Index](#) 

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

then extend your hand...

join us in...


Teach-Testing_→

An Ambitious Campaign by Tea-time with Testers

Cast Your Vote & Raise Your Voice !

It's definitely going to make the difference!

[Know More](#)

 **Click here To Cast Your Vote and Send Your Ideas!**

Subscribe here Right Away to get our all Issues for FREE

www.teatimewithtesters.com

Tea & Testing



with

Jerry Weinberg

The Golden Key (Part 1)

"One's first step in wisdom is to question everything—and one's last is to come to terms with everything."—Georg Christoph Lichtenberg

My Golden Key is a close companion to my Wisdom Box, because I cannot acquire wisdom without the risk of traveling to unexplored realms. The Golden Key represents my ability to open up new areas for learning and practicing, and also to close them if they don't fit for me at this time. Without my key, my consulting would become narrowly focused, or focused on areas in which I was no longer interested.



Nosy But Nice

For some reason, I've always had a fully functional Golden Key. Perhaps I was given this key by my father, who didn't pretend to have an answer to every question, but always asked me back, "How would we find that out?" Or, "Where would we go to get that information?" And sometimes we explored things together. In any case, if he did come up with an answer, he never just popped it out, whole. Instead, he always opened up his chain of exploration for my inspection.

Like my father, I'm a nosy guy, and I don't just mean the prominent proboscis I inherited from Harry. I investigate things I don't understand, and stop investigating when I'm no longer learning. Probably, that's why I've written so many books—researching a book is my standard excuse for exploring, or prying, or snooping.

Lots of my books started with a question—the best ones always did. Some actually have questions for titles—Are Your Lights On? and What Did You Say?.

Others had questions behind them, Golden Keys that opened doors for my mind:



- My doctoral research, Experiments in Problem Solving, asked, "Where do 'aha' experiences come from?"
 - An Introduction to General Systems Thinking asked, "What are the general laws of thought that apply to virtually every complex situation?"
 - The Psychology of Computer Programming posed the query, "What are the mental and emotional processes underlying the act of programming computers?"
 - When Don Gause and I wrote Exploring Requirements, we wanted to know, "How do we find out what people really want?"
 - And the four volumes of my Quality Software Management series are all based on the question, "How do managers affect the quality of software produced under their stewardship?"
- In fact, this article, itself, started with the question, "What are the most powerful tools that all successful consultants need?"

Polanski's Pointer

Well, writing's not for everyone, but there are other ways of using your Golden Key. Once I knew something about programming computers, I was often asked to help people find errors in their programs. At first, I didn't have much wisdom about "debugging," as this activity is sometimes called, and I wasted a lot of time following false clues.

Finally, one rainy December morning in the District of Columbia, my eyes were opened. We were working to a hard deadline—a scheduled rocket launch—and one of the programs just wouldn't work properly. Wally, one of the programmers, called on me to help, saying that they'd worked all night and hadn't located the problem. I asked him what they had already figured out.

"One thing I'm absolutely sure of," Wally said, "is that the bug can't be in the Red program. I've checked that one six times. And Sarah checked it, too."

So, taking him at his word, I plunged right into the Blue, the Green, and the Yellow programs—and never came out. That is, I didn't come out for lunch, and I didn't come out for dinner—both significant events in my working day. Finally, at around 9:30 in the evening, my stomach told me that Polanski's Deli next door was going to close in half an hour, so I took a break. When I got there, Polanski's crew had already cleaned up for closing, so I asked Julie, the counter waitress, for a take-out corned beef—extra lean.

"All our corned beef is extra lean," Julie insisted while assembling the sandwich.

"Hey, Polanski, bring me one of those take-out bags?"

"Harold must have put 'em away," Polanski shouted from the back. "Do you know where he put them?"

"No, but I'm sure they're not in the cookie cabinet. I already looked in there."



"Thanks," Polanski shouted back, and soon emerged from the kitchen proudly displaying a brown paper bag.

"Where'd you find it?" Julie asked. "I can never find stuff that Harold puts away."

"They were in the cookie cabinet."

I was dumbfounded. "Why did you look there?" I asked, "when she told you she was sure they weren't there."

"Precisely," said Polanski. "When Julie's that sure it's not there, it means that she believes it's not there, so she probably never looked there. So, it's probably there."

"Oh," I muttered. I grabbed my sandwich, paid the check, and rushed back to the office.

Wally was still studying the errant code. "Give me the Red listing," I insisted.

"Why?" Wally questioned, but handed me the listing anyway. "We know it's not there."

"Precisely," I said, and proceeded to find the bug in about two minutes.

And that's how I learned another way to use my Golden Key, a technique I call Polanski's Pointer:

If they're absolutely sure it's not there, it's probably there.

Polanski's Pointer tells me what doors to open, and a corollary tells me which ones not to bother with:

Don't bother looking where everyone is pointing.

After all, if they knew the right place to look, they wouldn't be asking a consultant to help them find it. And there's another version of Polanski's Pointer, one that I apply when I find myself "pointing" away from some subject. Whenever you believe that a subject has nothing for you, it probably has something for you.

Why? Well, if it's a subject, somebody is interested in it, so there's definitely something about it capable of arousing human interest. Therefore, if I don't see anything interesting about it, I must not even know enough to know why it can be interesting. That's a sure sign that I'll learn something if I open that closed door.

The Golden Lock

I have a trick for applying this personal version of Polanski's Pointer. I search for someone who is genuinely interested in the subject, then ask them for the one reference they would recommend to someone who knows nothing about the subject. This always works—unless I find someone who doesn't really love the subject, but is just making a living at it. There's a difference.

The reason there's a difference is that most people don't make full use of their Golden Key, and thus it's too easy for them to get stuck in a field that bores them. I call this phenomenon the Golden Lock:

I'd like to learn something new, but what I already know pays too well. The Golden Lock is a close cousin to the Golden Handcuffs corporations use to shackle their most valuable employees. But unlike the Handcuffs, the Lock is self-imposed, self-designed. Being self-designed, it's a far better trap than any Handcuffs could ever be, and only the Golden Key can unlock it. The "pay" for wearing the Golden Lock need not be money, though that's surely common among consultants. Quite frequently, the pay is prestige, or the envy of one's colleagues, or the gratitude of one's clients. Whatever the pay, it's not easily dispensed with—and thus the Lock.



That's why the Golden Key has two aspects—one that opens doors, and one that locks them again. I like to think my Golden Key is also very good at locking doors, but compared to my wife and partner, Dani, I'm a novice. Dani is particularly good at locking doors and moving on, having mastered several different areas of human knowledge in succession, and become a highly successful practitioner in each—teaching piano, doing anthropology, consulting to large organizations, and training professional dog trainers. Over the years, I believe I've learned Dani's secret rule, which I call Dani's Decider: When you stop learning new things, it's time to move on.

Dani's Decider is one of the most powerful secrets of consulting. Why? Consultants are hired for knowing what others don't know, so a consultant who stops learning soon decays in value. On the other hand, the less you know, the less likely you are to threaten your clients with change, so maybe you can become a "safe" consultant—one who offers no danger of changing the client's status quo.

Lock Language

We know that consultants can be threatening to their clients, especially if they're adept with their Golden Keys. That's why we often find our clients using "lock language" to keep us from opening their closets and seeing their real or imagined skeletons.

Some lock language is very direct. I've had clients invite me to examine their organizations, then tell me up front, "These are the things we don't want you to look at."

And, sometimes, when I apply Polanski's Pointer and say I want to look into X, they say, "No, I forbid you to look at X."

"Forbid" is rather direct lock language, but those with less authority tend to be more subtle. Possibly the most common lock phrase I hear in my work is "They won't like it if you ask about X."

Naturally, it's never very clear who "they" are, so I always counter with, "Oh, I didn't know that. Can you identify who 'they' are, so I can go ask their permission?" Generally, the speaker can't or won't identify a specific person, but if they do, I simply go to the person and tell them I'd like to ask about X.

An even more subtle way of locking doors is built into us by years of schooling— schooling that teaches many of us not to ask "too many" questions. Certainly I can understand why a teacher burdened with a large class of obstreperous children would want to restrict the number of questions per student, but these conditions don't apply to obstreperous consultants. So, when clients show non-verbal signs of impatience with my questions, I simply say, "Am I asking too many things all at once? I can come back if this is too much for now."

Of course, part of what makes my Golden Key golden is my skill in getting information that's behind locked doors—and getting it without provoking locking reactions in my clients. If I've done a good job of entering the client's system, I'm not likely to trigger any forbidding. Or, at least, I've avoided making contracts with clients

Whose locks are going to make it impossible for me to do what they're paying me for.

And, I've learned not to ask endless streams of questions. I don't need them, because I have so many other ways of getting information, as I've described in several of my books. So, I don't get much hard, direct forbidding, but if I'm not careful, my clients can lull my Golden Key to sleep.

Lullaby Language

Late one summer, I was called in to help an IT client learn to work better with their customers. I don't ordinarily travel in the summer, but this sounded like a real emergency, one where I had to be on the scene to calm down both parties. The customers were enraged with the IT manager because a new system wasn't ready on time, and IT manager was enraged with the customers because they hadn't delivered some essential information as promised, thus causing the entire project to lag its schedule by four months.

It was over 100 degrees outside, but even hotter inside—emotionally. Jeff, the IT manager, would smack the table and say, "You promised that the component pricing data would be in our hands by February first."

Penny, the catalog manager, would give him a steely-eyed glare and mutter, "We never promised that. Never!"

"Yes, you did!"

"No, we didn't."

... to be continued in Next Issue



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](#)

More Secrets of Consulting is another book by Jerry after his world famous book **Secrets of Consulting**.

This book throws light on many aspects, ways and tools that consultant needs.

“Ultimately, what you will discover as you read this book is that the tools to use are an exceptionally well tuned common sense, a focus on street smarts, a little bit of technical knowledge, and a whole lot of discernment”, says **Mr. Michael Larsen**.

More Secrets is definitely useful not only to consultants but to anyone for building up his/her own character by implementation of the tools mentioned in day to day life.

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

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

MORE SECRETS OF CONSULTING



Gerald M. Weinberg

TTWT Rating: ★★★★★

A photograph of a green, teardrop-shaped pendulum bob hanging from a thin wire. The bob is positioned directly above a circular pattern drawn in the sand. The sand is light-colored and has some small dark specks. The entire scene is framed by a dark blue border.

Speaking Tester's Mind

- straight from the author's desk



Six Impossible Things Before Breakfast

by Fiona Charles

Alice laughed. "There's no use trying," she said: "one CAN'T believe impossible things."

"I daresay you haven't had much practice," said the Queen. "When I was your age, I always did it for half-an-hour a day. Why, sometimes I've believed as many as six impossible things before breakfast..."

"Aha!" I thought, "Then you must have worked on software projects!"

Waiting for a bus a couple of years ago, I began listing software project lies. The exercise kept me nicely occupied until the bus arrived and through most of the ensuing ride. I ended up with an astonishing thirty categories of lies.

In case you think that sounds like an improbable number, I'll begin by outlining the parameters I set. First and foremost, I had personally to have heard the lie told one or more times. Each lie or category of lies had to be material to a software project, though it could have been told to make a sale before a project began or to describe a project after its end. Each lie had to be relatively common in the industry—or at least not rare. A lie had also to be significant to a project: to have influenced perceptions and/or decisions. And finally, I excluded malicious lies intended to subvert or sabotage an individual on a project, though I have both heard and (on one unhappy project) been the object of some of these kinds of lies.

Reviewing the list as I prepared to write this article, I immediately added a couple more.

The (depressing) reality is that in a career spanning three decades, I cannot recall a single project where there was not at least one significant and material lie. I thought I remembered consulting on one very nice-minded and squeaky-clean project, but then I recalled the programme manager telling a PM, "We're pushing the date of your project out, but it's vital that you do not tell your teams. Everyone needs to believe in the original date so they don't slack off."

Does the number of lies on my list horrify you? Am I exaggerating? Or could it be that we have all heard so many lies so often on software projects that we've become desensitized? I mentioned the "don't tell the team" lie to a notably honest programmer friend, and he said, "Well... we hear that one so often it almost doesn't qualify as a lie."

"Right", I said. "And how is it different from those other oldies but goodies:

'We're running 3 weeks behind, but it won't impact the end date.'

and

'We just need to work a couple of weekends to sort out all the quality problems.'"

(A project manager I once worked with used to say, "Show me where on your project plan it says 'A miracle happens here!'")

Perhaps you'll say team lies like this aren't deliberate lies—that the techies or the testers are just being over-optimistic, persuading themselves that the commitments they're making aren't patently impossible. We've all done it, haven't we? And yes, we probably have. But self-deception is still deception. A lie to one's self is no less a lie.

How far really are all those "we're going to make it" team lies from the infamous bait-and-switch scam some consulting firms routinely practice, or the deliberate underbids put forward to make a sale? ("We'll more than make it up in change requests," the sales people say.)

Lying of various kinds is quite common on software projects. Past the sale, the lies often continue with management arbitrarily slashing estimates and imposing upfront commitments to deliver fixed scope with fixed staffing within unachievable timeframes and budgets.

On projects that start out with fraudulent commitments, lies are propagated in the hotbed of fear. Managers demand certainty from people who are often barely in a position to give better than rough estimates, plus or minus fifty percent. Programmers and testers make desperate commitments they know they can't really achieve, and then, week after week, over-optimistically report their progress and status (lie). People lie—or avoid telling the truth, which is pretty much the same thing—to deflect blame and to get management off their backs, hoping to put off well-founded suspicions that they aren't going to deliver the impossible, and anxious to get on with productive work.

How many so-called "troubled" or "failed" projects would actually have gone much over time and over budget if they hadn't started out committed to fiction?

We may think lies like these are symptomatic of big waterfall projects, and indeed that is often true. But Agile projects are not immune to deception. A quick scan of the Agile blogosphere reveals plenty of discussion about impossible project or sprint commitments made by stakeholders outside the project teams or even by the teams themselves.

The rapid feedback built into an Agile process leaves much less room for practitioners to hide impossible estimates or falsify status. But teams calling themselves Agile have been known to play games with the concept of "done", redefining it to meet the actual state of the work when they haven't achieved their goals. Human nature is what it is, on Agile or waterfall projects.

It's a healthy sign, therefore, that two recent books by well-known and respected authors robustly address the topic of dishonesty on software projects and by software practitioners.

The Dark Side of Software Engineering: Evil on Computing Projects,ⁱ by Johann Rost and Robert L. Glass, is a survey of the bad things people do on software projects, plus some other software-related evils like hacking and computer scams. The range of subjects means the book is a bit of a hodgepodge, but it's a fascinating read. There's some numerical analysis I found less than compelling, mainly because the samples are too small for the numbers to have real statistical significance. This doesn't detract from the book overall, which has lots of good stories, qualitative analysis and suggested remedies from which we can learn important lessons. Among other benefits, you can learn to spot patterns of certain kinds of nefarious behaviour you may not previously have noticed on projects.

A nice counterpart to *Dark Side* is *The Clean Coder: A Code of Conduct for Professional Programmers*,ⁱⁱ by Robert C. Martin (Uncle Bob, as he's known in the Agile world.) I doubt anyone will be surprised to hear that Bob Martin comes out strongly against lying on software projects. You may be surprised at how he defines lying, including that it's a lie to say you'll try to meet a date when you already know you cannot achieve it. That advice alone is worth the price of the book. Martin also emphasizes the importance of learning to say "no"—an essential skill for people who want to tell the truth (one that Jerry Weinberg has been promoting and teaching for a long time).

Dark Side and *Clean Coder* are important for software practitioners of all specialities, including testers. The subject of professional ethics is vital for us all. We need to learn to recognize and stamp out lies and other ethical lapses on our projects—not just in other people, but in ourselves. These books will help. It's a bonus that they are also good reads: engagingly written and full of good stories we can relate to.

I don't know whether people on software projects are any more prone to dishonesty than the culture at large. I do know that deception in varying degrees is common on software projects. It's a dirty little secret we don't much explore, although it's an open secret in the business. I'm glad to see that other people are writing about it.

One of my articles on tester and consultant ethicsⁱⁱⁱ prompted a reader to protest that people can take grave risks telling the truth on software projects, and in a tough economy truth may be a luxury some can't afford. I believe that an expedient lie is the luxury we can't afford. Not for our professional reputations, not for our projects, not for our self respect. I think most testers would agree.

Lying hurts software projects. How many projects have you been on where "everyone knew" the schedule was fiction? Everyone except management, that is, the managers having conveniently forgotten they'd set the project up for failure at the beginning. And perhaps those managers had confidently told senior executives the fake schedule was all certain and wonderful—and now they're running out of budget and running scared. So they put pressure on their teams.

Apart from the cynicism engendered by living a lie, software people do shoddy work under pressure. Designing, coding and testing are all difficult work that requires a clear head. In my experience, the projects where people lie the most produce the worst software.

Lying hurts people too. Every time I present at a conference on "When a Tester is Asked to Lie",^{iv} one or two people take me aside and say, "This is so timely. It's happening for me right now and I don't know what to do." Others tell me it has already happened to them, and it's a nightmare they don't ever want to repeat. A test manager told me he'd been fired because "we don't think you're comfortable lying to the customer." ("Too right I'm not!", he said to me.)

Yes, it can be risky to tell the truth when others are lying. It can also be unexpectedly rewarding. I have more than once seen an unhappy project benefit from the act of a single tester or programmer

bravely stepping forward and saying, "I'm way behind. I'm not going to make the schedule, and I'd like to explain why." Sometimes that can be all that's needed to enable others to speak openly. Though the ensuing discussion might be painful, it could lead to a realistic replanning exercise that puts a project on an achievable path to recovery.

So why don't we just stop lying? We don't have to practice believing ANY impossible things before breakfast. We don't have to convince other people to believe them.

I've loved reading *Alice* most of my life, but I've never taken the White Queen to be a role model. Have you?

What lies have you heard on software projects? Add to my list.

I've written mostly in this article about schedule and status lies, but my list of thirty-plus includes many other types.

I've put it on my blog at <http://quality-intelligence.blogspot.com/> so you can see the whole list and add your comments. Can you add to the list?

Do you have experiences of project lies (or truth-tellings) you'd like to share?

Notes:

¹ Lewis Carroll, *Through the Looking-Glass* (Kindle edition), Chapter V.

² Johann Rost and Robert L. Glass, *The Dark Side of Software Engineering: Evil on Computing Projects* (IEEE Computer Society, John Wiley and Sons, 2011).

³ Robert C. Martin, *The Clean Coder: A Code of Conduct for Professional Programmers* (Prentice Hall, 2011).

⁴ I've published 4 other articles dealing with the subject of ethics, all on Stickyminds.com. Copies are also on the Publications page of my website www.quality-intelligence.com. Search for the titles:

- *Sophie's Choice* (September 1, 2007)
- *Deception and Self-Deception in Software Testing* (June 1, 2009)
- *Negative Positive* (February 8, 2010)
- *No Compromise* (June 21, 2010)

⁵ Besides the conference presentation "What Price the Truth: When a Tester is Asked to Lie", which deals with a specific type of project lying, I also lead an experiential workshop on the broader topic of "Deception and Self-Deception in Software Testing".

Biography



Fiona Charles teaches organizations to match their software testing to their business risks and opportunities. With 30+ years experience in software development and integration, she has managed testing and consulted on testing on many challenging projects for clients in retail, banking, financial services, health care, telecommunications and emergency services.

Throughout her career Fiona has advocated, designed, implemented, and taught pragmatic and humane practices to deliver software worth having—in even the most difficult project circumstances. Her articles on testing and test management appear frequently and she speaks and conducts experiential workshops at conferences. Fiona edited *The Gift of Time*, and guest-edited "Women of Influence", the January 2010 special issue of *Software Test & Performance* magazine. Fiona is co-founder and host of the Toronto Workshop on Software Testing.

She can be contacted on Twitter
@FionaCCharles

Back To Index

Esprit de corps: From Adversary to Ally

by Claire Moss

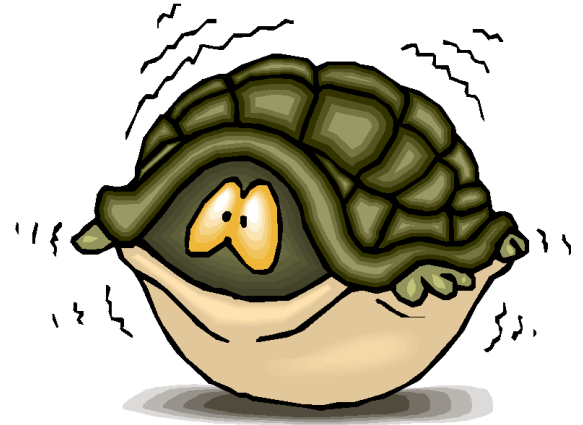
When I married a Civil Engineer, I should have considered that I was buying into devoting a portion of my conversation to storm water management. I also hadn't anticipated that driving down the road I would glance over to see that look on his face gazing out into the passing scenery, analyzing the concrete and asphalt. But then he should have been warned that I would repeatedly bemoan various retailers' credit card authorization systems that are not PCI compliant or otherwise print odd data on their credit card slips.

We analytical types engineer or otherwise, are an interesting mix. While we geek out about different subjects, we still have that passion to gather details and apply some rigor to the system. Our computer scientists and programmers also have their areas of absorption that diverge from our own. The great task before us is finding a way to connect despite those variations and to find - or perhaps build - common ground and avoiding the "us versus them" sentiment.



When I was younger, I was even more ignorant about what goes on under the hood of my car than I am now. After an unfortunate incident with family van, I had the good fortune to meet one mechanic who was purchasing the remains. He was very passionate about his area of expertise and excitedly expounded on the restoration and improvements he would make to the vehicle, teaching me enough to appreciate his excitement. Listening to him talk about the car produced an echo of his enthusiasm in me that I would never have expected. Although we didn't initially have a common interest, I responded to the joy in his expression and became engrossed in his plans and projects. It would be so easy to see this man as the exception to the rule. Instead, he has given me a lasting appreciation for car mechanics, who previously had been objects of suspicion when it came to recommending pricey courses of action for car maintenance. I finally had an exemplar to break down my impression of this group of people as "the other."

How do we move from that fear of the unknown to embracing the new and novel? Part of the fear is admitting the gap in our own knowledge. We are afraid that not knowing will lead others to think we are weak or unworthy. We need to let go of the compulsion to prove ourselves worthy adversaries and instead move toward becoming worthy apprentices. In my experience, software producers love to educate others, sometimes vociferously! When I approach my co-workers as a disciple to a master, they are often happy to elaborate on a variety of subjects, often ones that are new to me. Never underestimate the appeal of the uninitiated! As a developer, wouldn't you relish the opportunity to form a tester in your own image? When we absorb contrasting perspectives, we become better testers with a wider variety of ideas about how to pursue quality on the current project and in future projects. When we improve our technical understanding of the application under test, we ask better questions that might improve more than just the software; we may even influence the process.



One of the important things I work on every day is avoiding the "us versus them" sentiment that may once have been the norm between QA and developers. Although my development team has long been pseudo-Agile, I am not embedded and so there is the opportunity to view each team as "the other" and form opinions that way. Thankfully, I work with some great folks who want to produce high quality software and who don't want false assumptions to get in the way of that. We're all rather likeable as geeks go and try to play to each other's strengths. We choose to work together and we inspire each other to do the job well.

How do we encourage our developers to open up to us in this way? We want our team to want to produce high quality software and to avoid false assumptions that get in the way. We need to have personal connections that soften the edges of our professional roles. We need to have a genuine interest in the people around us so that we know them well enough to play to each other's strengths, encourage each other to take calculated risks, and compensate for each other's weaknesses, learning from our mistakes. We want to smooth any feathers ruffled by defect reporting and make our constructive criticism more palatable.

We can start with the basic facts that might be listed in a personnel file: gender, age, birth date, formal training, immediate family members, etc. Do we already have commonality here? People tend to be drawn to others like them, so we may already be ahead of the game. Can we remember events that matter to them (co-worker's birthday, wedding anniversary, welcoming a new child, loss of a pet)? Perhaps we empathize with their current state in life (starting a new job, empty nester,

experiencing health challenges). We want to be thoughtful about and toward the people with whom we spend so much time on the job.

From there, we can move on to look at their interests. Do we share any cultural shorthand? Or does each of us create new opportunities for growth? When we allow ourselves to disrupt the daily routine with less formal and more interactive pursuits, we produce opportunities to be vulnerable and to build camaraderie and trust. When we can laugh together, we can break the tension of taking ourselves and our roles so seriously. We can banter and bandy ideas about along with critiquing each other - and we must invite feedback from the developers on our testing! We need to keep each other honest about our needs so that we continue to work toward our shared purpose.



Gathering around the proverbial water cooler or eating together provide us invaluable chances to exchange thoughts, discuss the news of the day, clear the air of rumors, and many more benefits that come from simple conversation. The better we know our group members, the better we can serve each other. The most readily available source of information about our co-workers is often themselves. People tend to take advantage of openings to talk about themselves, so give them what they want. Ask questions about the things that occupy their minds. Get inside their heads and you can get a better feel for how they think about problems. Give yourself permission to venture outside your comfort zone. I welcome the opportunity to get out of my routine. That's usually the best way to get to know people. Civilly disagree. You will have a richer work life and might even end up with some lasting friendships that merit

keeping in touch even when someone moves on to a different job.

For a while, my developers and I had scheduled a Munchkin card game tournament over the course of many lunch breaks. After we burnt out on that game, many others followed, supplied by our resident board game and card game enthusiasts. For years now, we haven't run out of options. Though I tend to do well the first time through when they coach me enough to get the hang of things, I don't mind being trounced by more experienced players. Perhaps beating me at board games makes my constructive criticism more palatable? Either way, I don't have as many opportunities for gaming in my personal life as I once had, so I enjoy it.

I think of myself as a people collector. I can't help talking to whoever crosses my path. Some of those encounters change me for the better. I put in the time to show someone else that I value him or her and people respond to being appreciated and respected for who they are. At the end of the day, the social geek inside me relishes this merely for the fun of it all.



Feedback & Responses: May'11 Issue

Why Do People Happily Accept Poor Quality?

This question is important for testers to ask, and seek answers. James Christie has done a fine job of laying out a set of reasons based on Kakaonomics. Our job is not to change people's attitudes about quality. Our job is showing people exactly how some product behaves (as exactly as we can). If they don't care about quality, they'll ignore what we show them. On the other hand, some people care too much about quality as they define it. We don't need people pushing testers.

Testing and Management Mistakes: Causes

The previous article has a blaming flavor to it, but in this article, Magnus the Manager's behavior seems more super reasonable than blaming. In any case, it's not taking Tim into account at all. And, of course, Tim hasn't helped the situation by placating. But Tim's behavior is not unreasonable, because Magnus has power over him, so Tim fears what Magnus might do to him. Markus Gartner's article makes a strong case for why we need better managers in Testing.

The Joke's On You

Nathalie has given us some lightness, and argued for lightening up in testing once in a while. It's a good lesson and a fun read. I hope we have more jokes with morals. Here's my contribution, the classic of all testing jokes (with at least two endings):

A cop sees a drunk on his knees under a streetlight and asks him what he's doing there. "I'm looking for my keys," says the drunk.

The cop kneels down and helps him search, but finds no keys. "Are you sure you dropped them here?" "Oh, no," says the drunk, pointing up the street. "I dropped them there."

"Then why are you looking here?"

ENDING 1: "Because the light's better here."

ENDING 2: "Because this is where I found them last time I lost them."

If you can't see why this is a testing joke, you're in the wrong business.

Test Cases in Agile a Waste of Time ?

This is a terrific article about Petteri Lyytinen's philosophy of testing. It could be improved in two ways:

- a more descriptive title

- at least one example of what he writes down as the result of his Iterative Test Development.

- Jerry Weinberg

Biography



Claire Moss has been testing software for over 7 years. Although authoring a testing blog and article are new for her, Claire has always had a passion for writing, which might be a strange trait for a Discrete mathematician. After working briefly as a software programmer during college, Claire signed on as a quality engineer after graduation.

When you find your calling, you never look back! Claire continues to use her evil powers for good on the job and on her blog: <http://blog.aclaification.com>

Claire can be contacted on Twitter @aclaification

Hi,

James Bach recommended your magazine at his Rapid Software Testing course in Melbourne which I completed today. I look forward to reading your insights and contributing where possible.

Best wishes,

Suzan Moses,
Melbourne, Australia



Dear Suzan,

Thanks for writing us. Getting recommended by Great Testers in the community is our biggest honor and we consider it as a receipt of the hard work that we are doing.

Feel free to send us your article. We would like to hear and publish your thoughts around Testing.

Sincere Regards,
Editor

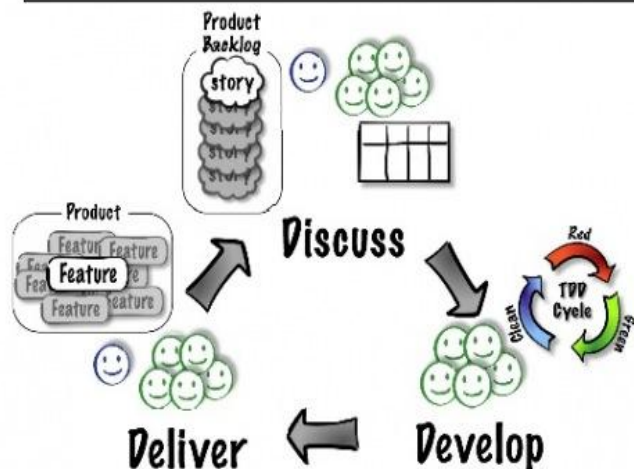


Does ATDD=Waterfall ?

by Adam Yuret

I realize I go on incessantly about what a revelation Elisabeth Hendrickson's "Introduction to Acceptance Test Driven Development" class was for me. I wanted to write about areas where I've seen opportunities to benefit from implement it. But first I'd like to give some background of what I learned from this course.

Acceptance-Test Driven Development (ATDD) Cycle



(Model developed with Pekka Klärck, Bas Vodde, and Craig Larman.)

Copyright © 2010 Quality Tree Software, Inc.

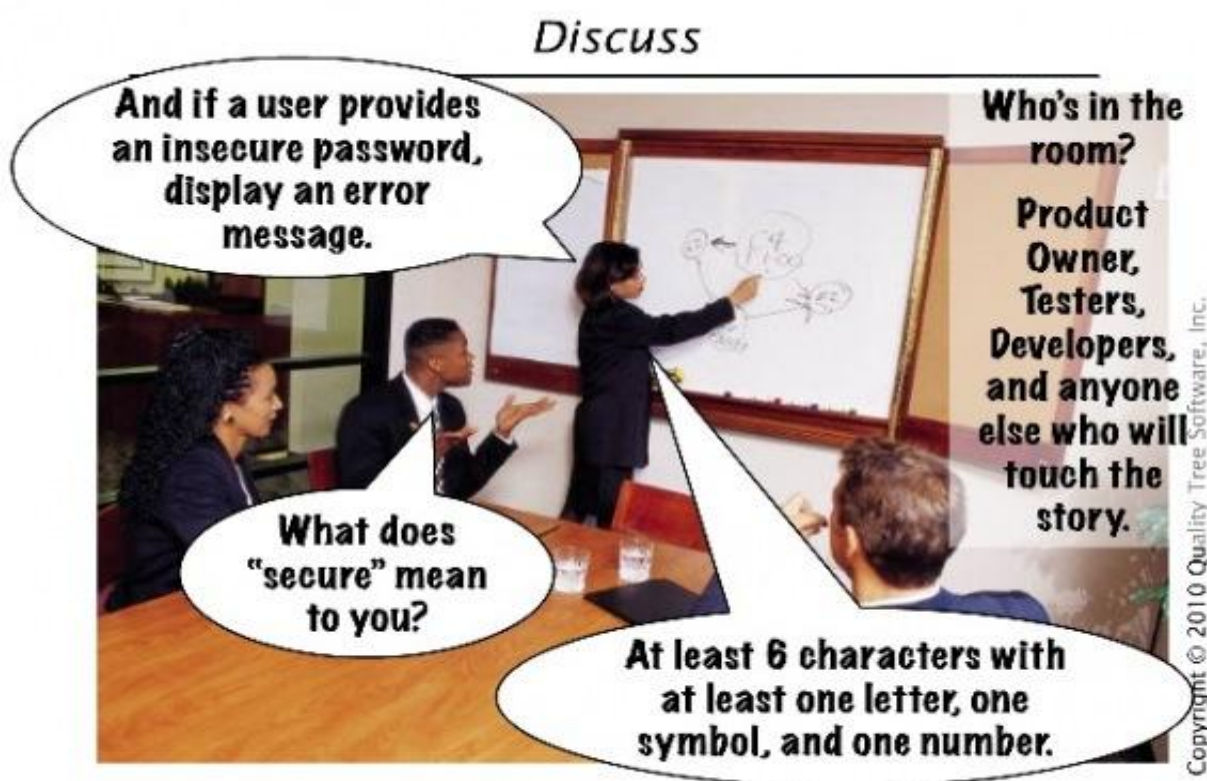
In a nutshell ATDD (Acceptance Test Driven Development) involves bringing every team member that will touch a user story together at the beginning of a sprint to discuss the story to outline acceptance criteria and attain a shared understanding of the stakeholder's vision.

There is a lot more to ATDD to discuss however for the sake of brevity I'd like to focus on the aspect that excited me the most, story workshops.

In a story workshop every team member who will touch the user story is gathered to discuss details of the story. One person facilitates the discussion, and after the story is described, testers and programmers ask questions and share concerns with the stakeholder. The end goal is to have shared understanding through examples of the stakeholder's expectations. Programmers might ask if an implementation strategy is acceptable, testers almost certainly will discover areas of risk in the story's description.

For the purposes of this class Elisabeth ran us through a few mock story workshops during which Dan Snell volunteered to play the part of the stakeholder. This was a brilliant way to teach by doing. We uncovered many interesting aspects of the story workshop that were best illustrated by allowing the group to collaborate on this mock story.

Some of the things we uncovered may seem rudimentary and obvious. When defining acceptance criteria it's important not to take these important details for granted. Other things were less predictable. As testers in the room began to come up with test cases Elisabeth wrote them down on her flip chart. Some test cases were too deep to merit being included in basic acceptance. But mostly the test cases were simple. It may seem like a small thing to stand up and say "If the user inputs a date which occurs in the past, and it still posts a listing, this story fails acceptance." until this seemingly obvious assumption is immediately shot down by the stakeholder in the room. She might say "I never thought to write that case into the story, but now that you mention it, I want it to post outdated listings."



Maybe adding this feature is so involved that the engineer in the room pipes up and says "If you want me to code something that will allow an expired listing to be created it will conflict with existing code and require a significant refactor." Maybe the stakeholder demands this feature. She can most certainly have it, but the effort to create and test it will drive it well out of scope for the story. In this case a new story is born and we move on to wrapping up the story in question.

One of the testers in the front row actually exclaimed in mock-frustration "I don't care it's *your* product!"

This simple story resulted in a lot of discussion. Elisabeth was flipping that flip chart and taping paper to the wall as quickly as we were defining criteria. In the end we felt like we had achieved a shared understanding of what the stakeholder wanted to get from the story. Every boundary we could think of at the time was addressed and accepted or rejected by the stakeholder. This didn't mean the story was chiseled in stone, but I believe by having the story workshop we significantly reduced the odds that changes would be needed. Or if changes were needed it would not be due to a lack of communication. I would imagine that this understanding would reduce the amount of discussion necessary later when changes are needed.

During sprints I see a need for this kind of shared understanding. I find issues when testing that are not bugs but might be areas that might confuse a user.

I always take these concerns to the stakeholder and sometimes they tell me they are perfectly happy with the story the way it is. But sometimes they decide to accept my suggestion and make a change. When that happens the dev has to double his effort. He now has to remove what he did and replace it with the change. I think this is where a team pays the cost of not having story workshops.



Alternatively a programmer while trying to implement something defined in a wire frame or a user story might discover something unforeseen that requires diverting from the defined criteria. I see this sometimes in story notes (e.g. "This story is ready to be tested, as per my discussion with (the programmer) these changes to the user story were made.")

Of course it's hard to quantify that cost. Story workshops are definitely expensive. A complex story workshop can take hours to wrap up. Lean/agile methodologies abhor rigorous documentation and

heavy process. This can make story workshops a tough sell. How does one explain that we're not proposing to create a tome of specifications for each story. We're not trying to revert to a waterfall process. We're not slowing the process down with this story workshop, ultimately we're trying to speed it up. Much like test driven development it seems like a lot of overhead at the start of the project but I suspect, like TDD, the end result can be increased throughput and improved quality.

There is a lot more to ATDD than the story workshop I've described here and as I learn more about those aspects I'll share them in this blog. In 10 days or so I'll be taking what I've learned and holding my first mock story workshop with my product and development teams. Hopefully, despite having taken her class way back in October, I do Elisabeth justice. I am eager to hear other people's experiences with this process and thoughts about this post. Please comment and discuss, as I still only have an academic understanding of ATDD. I am very eager to hear others' real world experiences.

Biography

After 8 years at WebTrends testing an enterprise level SaaS data warehousing product which included building and maintaining a large scale testing environment, Adam currently works as an "army of one" tester for VolunteerMatch. VolunteerMatch is a national nonprofit organization dedicated to strengthening communities by making it easier for good people and good causes to connect.

Adam is a relative newcomer to the context Driven community and is currently working to build a testing process for a project that is transitioning to an agile/scrum methodology.



Adam has recently made a presentation at QASIG (March 2011). Its video streaming can be viewed [here](#). He can be contacted at adam.yuret@gmail.com or on twitter [@AdamYuret](https://twitter.com/AdamYuret).

A Voice on "Teach-Testing" Campaign!

Hello Tea-time ,

Here in Brazil the Education is the same. We don't have Software Testing as a separated course. We don't learn Software Testing in Engineering Courses. So, we need to accomplish these courses and begin with MBA or Postgraduate.

We could have separated subjects on these courses, preparing us to the world, instigating to learn more about testing and bringing out better professionals thereby.

Thank you for the magazine and this initiative!

- João Paulo Percy, Brazil

Back To Index



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

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

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

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

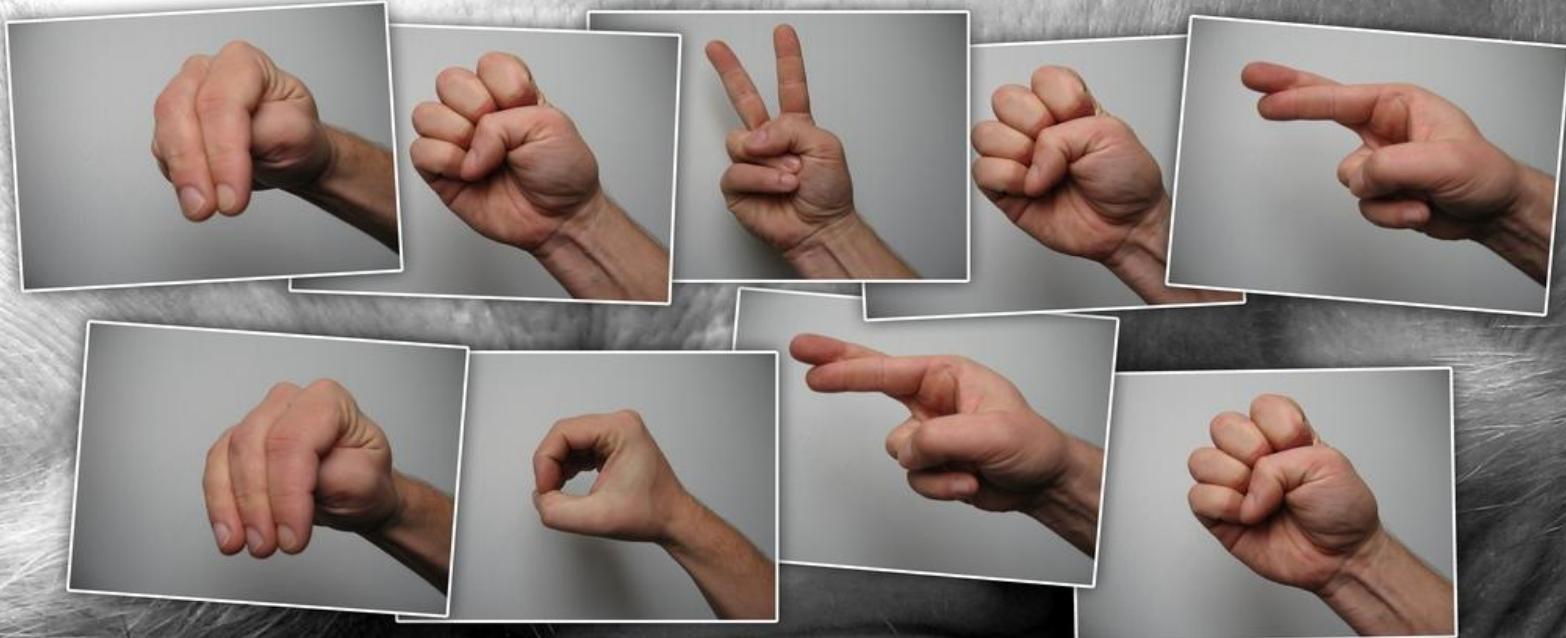
Make sure to write **Feedback for <Article Name>** in your Subject Line.



A photograph of several young students in a classroom, seen from behind, with their hands raised in the air. They are facing a chalkboard that has some faint writing on it. The students are wearing colorful shirts: light blue, red, orange, and green. The entire image is framed by a thick black border.

In the school of Testing

for your better learning & sharing experience



Learning to Communicate Better with Programmers

image: assbach

by Lisa Crispin

Recently, my team decided we needed another tester. We discussed what qualifications we'd like to see in a. The programmers on our team said they wanted a tester with some programming experience. This puzzled me, because on our team, though I and the other tester have a programming background, the programmers do most of the test automation. Why do we need a tester who understands programming? The programmers thought about this for a long time. Then they said, "We can communicate so much better with a tester who understands programming concepts".

Personally, I would be happy to hire a tester who is good at collaborating with customers to understand requirements, and who is an awesome exploratory tester. But I understand my programmer teammates' viewpoint. We use the Whole Team approach to software development, including testing. Everyone on the team takes responsibility for making sure all testing activities are completed for each user story. This requires intense collaboration. If a programmer feels that a tester doesn't "speak her language", that creates a barrier.



Communication between testers and programmers is a two-way street. Since this article is in the **Tea-time with Testers**, I'm going to focus on what testers can do to enable better communication with programmers. What can we testers learn in order to communicate more effectively with the programmers on our development team? Here are some suggestions based on my experience.

Learn Programming Concepts

A common vocabulary always helps. I was a programmer back in the structured programming days. I also programmed in a 4GL. This experience is useful, but I still had to learn object-oriented (OO) concepts. I taught myself Ruby with the help of books and a kind teammate. Though our application is written in Java, experience with coding Ruby helps me understand design discussions.



Years ago, I worked in a company where the QA group was separate from the programming group, and they didn't want to help us with test automation. The application was written in TCL, and I heard it was a good language for scripting automated tests as well. I got a book and taught myself some TCL, and started automating tests with it. Whenever I had a problem, the TCL programmers were happy to help me with it.

If you have no programming experience, where should you start? There are so many resources, but here's what I suggest. Get *Everyday Scripting with Ruby: For Teams, Testers and You* (Brian Marick, 2006). Work through all the examples, and you'll develop competency with Ruby, OO concepts, and programming terminology. For some introductory object-oriented design knowledge, check out this article by Dave Thomas and Andy Hunt, http://media.pragprog.com/articles/may_04_oo1.pdf.

Integrated Development Environment

There are many different IDEs, and plenty of personal preferences as to which are the best. I highly recommend that you ask for the same IDE that is used by the programmers on your team. Install it, and ask for help setting up the project for your team's source code. You're not going to write production code, but you can use the IDE to *look* at the source code and (hopefully) unit tests. If you do any test automation, such as GUI tests, you can take advantage of your IDE's features.

My first venture into an IDE was with Eclipse. At the time, a couple of programmers on my team used it, and they helped me set up a project for the production and test source code. However, everyone on my team eventually moved to IntelliJ Idea. This seemed harder to use at first, but here's the news – if you are using the same tools as the rest of your development team, they're much more likely to help you. Because I use an IDE, I can look at production code and even ask intelligent questions about it. Ask a teammate to help you install an IDE and set up your project.

Database

Over my career, I've always found that understanding the database design is essential, not only for communicating with other development team members, but to do a good job of testing. I spent years testing database software, so I'm familiar with many different ones. Your team most likely uses some relational database such as Oracle or MySQL. There are tools that can help you navigate the database tables even if you don't have much SQL experience. For example, I use SQL Developer to retrieve information from our Oracle database.

SQL is indispensable if your application works with a relational database, and it's easy to learn simple SQL queries. Any motivated tester can develop competency in SQL quite rapidly. We hired a tester who didn't know SQL or Unix, but since he was teaching himself Ruby, we trusted that he was motivated to learn them quickly, and we were right. If you need some grounding in SQL, check out online tutorials such as <http://www.sqlcourse.com/index.html> and <http://www.w3schools.com/sql/default.asp>.

Maintaining Test Environments

Large companies have entire organizations devoted to maintaining all the development and test environments. In smaller ones, such as my own, testers need the skills to maintain their own test environments. Depending on your situation, this could mean Windows, Unix or who-knows-what. Your willingness to maintain your own test environment, and control what gets deployed there, is the most important factor. Learning some basic administrator commands isn't difficult. You can find plenty of introductory tutorials online, such as <http://www.ee.surrey.ac.uk/Teaching/Unix/>. There are also classic books including *Unix in a Nutshell* (Arnold Robbins, O'Reilly, 2005)

Domain Knowledge

We can't be effective testers without understanding the domain. Programmers need to know the domain too, but they are so focused on getting each piece of functionality to work, they often lack time to see the larger view. Testers tend to be "big picture" people. Spend time understanding all the business problems, and you'll be able to add much more value to your programmers' efforts.

Be a Valuable Tester

If you can collaborate well with programmers on your team, you have a big advantage over testers who work in isolation. Make time to understand basic programming concepts and your product's overall architecture. Learn skills that let you manage your own test environments. Learn your business domain so you can help your teammates provide the right business value.

Naturally, you don't forget your valuable testing skills. The programmers on your team can help ensure you have time for exploratory testing by ensuring adequate automated regression test coverage. You'll help the programmers both by finding issues in their code, and by helping them to better understand customer needs. When the whole team works to ensure all testing activities are planned and executed, we all enjoy our work more and can be proud of the software we deliver.


Biography



Lisa Crispin is the co-author, with Janet Gregory, of *Agile Testing: A Practical Guide for Testers and Agile Teams* (Addison-Wesley, 2009), co-author with Tip House of *Extreme Testing* (Addison-Wesley, 2002) and a contributor to *Beautiful Testing* (O'Reilly, 2009). She has worked as a tester on agile teams for the past ten years, and enjoys sharing her experiences via writing, presenting, teaching and participating in agile testing communities around the world.

Lisa was named one of the 13 Women of Influence in testing by Software Test & Performance magazine. For more about Lisa's work, visit www.lisacrispin.com.

Lisa can be contacted on Twitter @lisacrispin.



Increasing test effectiveness with affordable custom tools

by David Vydra

"Imagination is more important than knowledge. For knowledge is limited to all we now know and understand, while imagination embraces the entire world, and all there ever will be to know and understand."—Albert Einstein

As with any profession, testing has both a fun and tedious side. Creating test strategies and exploratory testing are very enjoyable and intellectually challenging activities, while doing manual setup and manually executing scripts are not. Most of the test automation efforts have focused on automating test scripts and automatically checking the results, but there are many other useful instances of automation.

In this article I present several examples from my tool smything experience. While none of them may be directly applicable to your situation, I hope that they will inspire you to look for opportunities where custom-built tools can make your work more enjoyable.

Speeding up Sanity Tests

At one of my engagements I observed the QA team do a fair amount (about two person days worth) of scripted manual testing for each patch and minor release. This sanity testing effort happened every three weeks on average. When I asked about the effectiveness of this exercise I was told that the last time any bug was found using this testing process was about two years ago. This seemed rather expensive, but the team was adamant about keeping this process because the memories of delivering buggy patches were still fresh in their mind.

This manual testing was in addition to many thousands of automatic tests that ran for each build, but the key was that testers wanted to see the software work with their own eyes. The solution we implemented consisted of Selenium/WebDriver automation of all the manual scripts -- we only automated the steps, not the results checks. For each transition in the UI we took a snapshot of the

screen and wrote it into a file organized in such a way that all of the snapshots for each test were in a separate directory.

This greatly sped up the sanity testing for each release; all we had to do was to look through the snapshots to convince ourselves that no serious badness had occurred. The time we saved was put to better use by performing exploratory testing based on the specific changes documented for each release.

Improving Reports Testing

Later that year I was assigned to test the reports feature. The setup was as follows: reports were produced via a third-party tool and ran in a separate server that talked directly to the database server. There was only one Report Analyst for the entire company — needless to say he was a very busy guy. Setting up the reports server when the reports configuration changed was tedious and error prone, so testers always put off reports testing to the end of the release cycle. Thus reports were always late, dreaded by testers and often had to miss the main product release and shipped separately.

I started to look for opportunities to apply automation to fix some of these pains. First I looked at automating the checking of the final report. This was a non-starter because the report UI was using copious amounts of custom JavaScript and was not amenable to web automation tools such as Selenium/WebDriver.

To establish testing priority, I examined the history of the testing effort in this area which has been ongoing for several years. It was obvious that most of the bugs were related to schema drift, but there was no easy way to determine which reports were affected by schema changes because report templates used a custom format to represent the data model.

Then it hit me: the reports server probably has diagnostic logging capability to trace the SQL that it sends to the DB server. Sure enough, after a quick search through docs, I was able to see the generated SQL.

Next I ran each report by hand to capture the generated SQL for the most common scenarios. With SQL in hand, I created plain old JUnit tests whose sole purpose was to break when the DB schema changed. These tests ran in a few seconds and I was allowed to check them into the main continuous integration server suite. Now when one of these tests broke, we could disable it and file a bug against the reports component.

Failed tests notified the Reports Analyst about upcoming work much earlier in the cycle and were able to schedule his time much better. On the testing side, I wrote a script to deploy the reports server using a single shell command. With the tedious and error-prone setup eliminated, testers were happy to do exploratory testing as soon as the new reports were checked in. When the bug was fixed, we replaced the SQL string in the JUnit test and re-enabled it.

Fixing the Build

This last example is perhaps the most bizarre. I started working as a QA engineer for a company that produced an enterprise product where the user interface was heavily customized by the customers. The UI source code was shipped along with the product. Half of the customers were using Java; the other half were using C#. When I arrived, the team was using a free Java to C# translator to write the system in only one language. This particular translator was not designed for repetitive use in a

continuous integration environment. It did not even produce C# that was 100% correct and, after each translation cycle, an engineer had to manually fix the missing parts.

The translator took about six hours with our code and then the engineer usually spent a few more hours. Needless to say this approach could not scale and was not Agile at all. This state of affairs really bothered me and I wanted to find a fix, but writing compilers was not part of my background then. Luckily I have an outgoing personality and I am not afraid to ask for help from other engineers. Several solutions were offered but seem too expensive to implement for our budget.

Then one day over lunch an engineer suggested that we try the java compiler that is part of the tools.jar library that comes for free with the Java language. This was our breakthrough as we learned that it contained at least 70 percent of our solution and, most importantly, it was rock solid, having been tested by thousands of developers over many years.

With this approach we were able to finish the project in about four months and translate our entire code base in under four minutes with 100 percent accuracy. The build was fast again and we could get back to our normal work.

If you are a tester and you notice that you are doing boring and repetitive work, or the system is hard to setup for testing, don't despair. Talk to your programmers (and managers where appropriate). Programmers love to solve problems and chances are good that you will get your tools built in short order.

Testing should be a fun and creative activity. Demand it!

Biography



David Vydra has been building software since 1987 and has been a practitioner of the Agile style since 1998. He has been an application developer, tester and tools builder with some of the top companies in the Bay Area.

His current interests focus on Agile Testing, Executable Examples, Developer/QA collaboration and building tools to support complex continuous integration scenarios and exploratory testing.

David's blog is <http://testdriven.com> & he can be reached on Twitter @vydra .



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

Jump on to Page 51

Back To Index



Testing...



with Anurag

Mobile Apps Testing: Need, Challenges and Opportunities

Who is Anurag?



Anurag Khode is a Passionate Mobile Application test engineer working for Mobile Apps Quality since more than 4 years.

He is the writer of the famous blog **Mobile App Testing** and founder of dedicated mobile software testing community **Mobile QA Zone**.

His work in Mobile Application testing has been well appreciated by **Software testing professionals** and **UTI** (Unified Testing Initiative, nonprofit organization working for Quality Standards for Mobile Application with members as Nokia, Oracle, Orange, AT & T, LG Samsung, and Motorola).

Having started with this column he is also a Core Team Member of **Tea-time with Testers**.

The world today can surely experience the boom of Mobile arena. From feature phones to Smart phones mobile handsets are entering into market like never before.

According to Gartner, 428 Million Mobile Communication Devices Sold Worldwide in First Quarter 2011, a 19 Percent Increase Year-on-Year. Manufacturers, Carrier Platforms are coming up with their own mobile application stores with hell of free and paid application. From mobile games to business applications and from entertainment to LBS applications mobile application stores are having applications from all horizons.

Over 300,000 mobile apps have been developed in last three years and apps have been downloaded 10.9 billion times. But with the opportunities in this market the competition is also getting much tougher day by day. Just to add to the fact that 1 in 4 mobile apps once downloaded are never used again. So there is no doubt that in order survive in this market, Mobile Developers needs to come up with good ideas and superior quality of Mobile applications and thus, here **"Mobile Application Testing"** comes in the picture. Mobile Application Testing is the new area in the field of testing. Talking about the challenges I would like to emphasize that Mobile Application Testing challenges are somewhat interrelated but little different for Mobile Developers and those for Mobile Testers. However before indulging directly in to Mobile Application testing we should know the real challenges which community of Mobile developers is facing.

Challenges in Mobile Application Testing for Mobile Developers/Industry:-

- 1) **Thousands of Mobile Handsets:** - For mobile developer it is one of the biggest challenges that he may ever face. In order to make a mobile application, one needs to be very sure about the devices he/she is targeting. According to a research, 1388.2 million handsets were sold in year 2010. These devices are of different screen sizes, input methods (QWERTY, touch, normal) with different hardware capabilities. Knowing the fact that testing on every device is not possible and feasible at all, the diversity in handsets is a big challenge for Mobile developers.
- 2) **Different Mobile Platforms/OS:** - There are different Platforms/OS currently present in the market. Android, IOS, BREW, BREWMP, Symbian, Windows 7, Blackberry (RIM) and so on. Diversity in mobile platforms, different OS versions, and platform limitations make it a bit difficult and challenging for developing mobile apps and so for testing them. There might be chance of inconsistency in terms of functionality across multiple devices of same platform and every platform may have some limitations.
- 3) **Different Mobile Carriers/Manufacturers:-** There are different mobile carriers in the market and every manufacturer may have some norms for the mobile applications if that application is coming preinstalled on the device. Verizon wireless, AT & T, T-Mobile, Orange, Docomo, Airtel, Vodafone, Reliance are some of the carriers. Please note that Mobile developers here means entity (may be any organization) or a person developing the mobile application and not mere the software developers.

I hope now you are clear about the need of Mobile Application testing and the real challenge this industry s is facing for Mobile Application Testing. Here I have just tried to give insight of Mobile Industry growth and Industry challenges as per Mobile Application is Concerned. As now I am associated with "Tea Time with Testers" we will be exploring all the area of Mobile Application Testing from Mobile Apps Testing Challenges for QA'S to Mobile Testing Strategies, Mobile Tools and Utilities, Mobile Automation Tools and much more. We will not only explore testing aspects for mobile apps but will also keep look in to market trends and scope and role of Mobile Testers in to them.

Just to precise Mobile Application Testing market has a great potential and it can be considered as one of the most immersing field/domain in Software Testing. The more you dig the more it will come out with something. Due to dynamic nature of Mobile Applications and small release cycle of mobile apps there are many challenges for Mobile Testers also and we will take a look in to this in my upcoming articles in **Tea-time with Testers**. Till then Stay tuned and have nice Tea-time ☺ !

testing intelligence

- *its all about becoming an intelligent tester*



an exclusive series by **Joel Montvelisky**

Testers !! Know your Business !

In April issue of *Tea-time with Testers*, I wrote an article titled *Principles of good bug reporting*.

It talked about the basic things that make a good bug report and helps testers not to make the typical mistakes that will be criticized by other members in their team. One of the points I mentioned is to always make sure you are giving bugs the correct Severity & Priority grades, since it is typical of inexperienced testers to give their bugs higher severity with the (sometimes unconscious) aim of having them corrected by the development team.

Jerry Weinberg commented about this article saying among other things "***I don't think testers should be prioritizing bugs***"; and when I look back at what I wrote I agree with him, I was only half right on asking tester to set their severity and priority grades correctly.

A tester should know his business

The half that I was right was that a tester should be able to set the Severity of his bug correctly.

Severity should be (as much as possible) an objective measure of the damage caused by the specific bug to the Application Under Test (AUT). And it is something a tester should be able to determine correctly.

The best way for testers to set the severity of a bug correctly is by understanding enough of his product.

- What does the product do (and not only what is written in our site or in the marketing brochures)?
- Who uses the product and under what circumstances?
- What are the different flows a user can work with the product?
- How knowledgeable are the users in order to find workarounds to specific bugs?

And all the rest of the non-trivial information about the product and its users that will allow him to determine correctly how severe is a specific bug in the system.

I expect every tester, within a logical amount of time and with the correct guidance and information from his team, to reach this level of knowledge. And the reason she needs to know the product this way is not only in order to set the severity right, this is the only way by which this person will be able to define, design and execute his tests correctly.

A tester should also know NOT to do the work of others in his team

On the other hand, and here is where the criticism from Jerry came, we should not require of a tester to determine the priority of a bug, remembering that priority is related to the time when a bug should be fixed, and it is only partially determined by its severity.

Let's understand first of all that priority unlike severity is a subjective measure, subjective because it will be determined based on many personal (or team-related) factors and so it cannot be giving outside the context of the whole team.

What factors may affect the priority of a bug:

- Obviously one of the most important factors is the **severity**. We will want to fix critical bugs before we solve the less critical bugs.

But there are many other factors such as:

- **Complexity of the fix**. We will want to solve complex fixes before we solve the trivial ones. We do this in order to assure that a complex fix will be tested more and more thoroughly. On the other hand sometimes we will choose not to solve a bug because there is not enough time to test it enough and so we prefer to release the bug as is than to solve it and introduce more severe bugs or delay the release of the version.

- **Available resources.** Sometimes we choose to solve bugs based on the developers who are available, for example solve a simple GUI bug now and leave the complex DB bug for later due to the fact that the GUI developer is leaving on vacation next week, or because we need to send the screens now to the customer for validation.

- **Bug fix clustering.** It is logical that if we are going to be working on a specific area of the code we try and fix as many bugs in it as possible, and so sometimes we will give bugs priority based on where we are already fixing other bugs.

- **Earlier promises.** This is one of the most problematic, but also very easy to understand... We will want to fix first the bugs we already promised to customers, since they are already aware of them and they are expecting them to be fixed. So if we have 2 bugs with the same severity or even a bug that has a lower severity but was already promised to be fixed to a paying customer, we will most probably be taking care of it before we handle other bugs.

As you see, there are many reasons (all of them good and sensible) why we cannot set the priority of a bug, and why we should leave this to either the person who has all the information or at least make sure the decision is taken by the whole team in order to reflect as many of these points (and also the ones I left out) into consideration.

Biography

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

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

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

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

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



A group of managers were given the assignment to measure the height of a flagpole. So they go out to the flagpole with ladders and tape measures, and they're falling off the ladders, dropping the tape measures - the whole thing is just a mess.

A tester comes along and sees what they're trying to do, walks over, pulls the flagpole out of the ground, lays it flat, measures it from end to end, gives the measurement to one of the managers and walks away.

After the tester has gone, one manager turns to another and laughs. "Isn't that just like a tester, we're looking for the height and he gives us the length."

Back To Index





Call for Articles !

Have you got something to say?

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

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

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

Submit your articles, stories, thoughts around software testing.

now its your chance to be heard...!

T ' Talks



T. Ashok exclusively on software testing

Landscaping: A Technique to Aid Understanding

In my last article (The diagnosis) Joe faced with the problem of understanding a new application in a domain that he is unfamiliar with, has the "Aha" moment at the doctor's office during the process of 'diagnosis'. He sees parallels in doctor's questioning technique to diagnose the problem and his problem of understanding of the application. He understands that decomposing the problem into information elements and establishing the connections between these elements enables him to come up with good questions to understand the application. Voila!

Joe figured that understanding an application is not just about walking through the various features via the user interface (assuming that the application has an UI), it requires a scientific/systematic walkthrough of various elements commencing from the customer's needs/expectations and then into the application's deployment environment, architecture, features, behavior and structure.

This is a technique called Landscaping, a core concept in HBT (Hypothesis Based Testing) that enables one to systematically come with meaningful questions to understand the end users, application and the context. It is based on the simple principle "Good questions matter more than the answers". Even if questions do not yield answers, it is ok, as it is even more important to know what you do not know.



Now onto Landscaping in detail...

The objective of testing is to ensure that the product or application meets the expectations of the various end users. That said, how does one get into the mind of the end-users to be able to appreciate expectations that they have? Let us dissect the problem...

Firstly start from the outside - the marketplace and various types of customers in the marketplace where the application is going to be used. Then identify the various types of end users (or actor) in each type of customer in the marketplace. Now identify the various use cases/requirements for each user type and then the technical features that constitute the business use case or requirement. Now viewing each requirement from the perspective of the end user, identify the attributes that matter for each requirement/feature and then ensure that they are testable.

Then move inwards. Understand the deployment environment, structure of the application (architecture) and the technologies used. Moving further in, identify who uses what feature, the profile of usage, ranking of a feature/requirement in terms of business importance and then the conditions that govern the feature. Finally understand how each feature/requirement affects the other features/ requirements and then understand the state of a feature (new, modified, status quo).

Let's picturize this...



What are we trying to do? We are trying to create a mindmap that consists of various information elements, then question to understand the information elements and more importantly establish the connections between the various elements. The act of understanding the individual elements and their connections results in questions that aid understanding. As we proceed, we establish a clear baseline of end user types, requirements/features, attributes, environment, usage profile...

Let us look at some of the questions that emanate by applying Landscaping...

Marketplace	What market place is my system addressing? Why am I building this application? What problem is attempting to solve? What are the success factors?
Customer type	Are there different categories of customers in each marketplace? How do I classify them? How are their needs different/unique?
End user (Actor)	Who are the various types of end users (actors) in each type of customer? What is the typical/max. number of end-users for each type? Note: An end user is not necessarily a physical end user, a better word is 'actor'
Requirement (Use case)	For each end user, what do they want? What are the business use cases for each type of end user? How important is this to an end user - what is the ranking of a requirement/feature?
Attributes	What attributes are key for a feature/requirement to be successful (for an end user of each type of customer)? How can I quantify the attribute i.e. make it testable?
Feature	What are the (technical) features that make up a requirement (use-case)? What is the ranking of these? What attributes are key for a successful feature implementation? How may a feature/requirement affect other feature(s)/requirement(s)?
Deployment environment	What does the deployment environment/architecture look like? What are the various HW/SW that make up the environment? Is my application co-located with other applications? What other softwares does my application connect/inter-operate with? What information do I have to migrate from existing system(s)? Volume, Types etc.

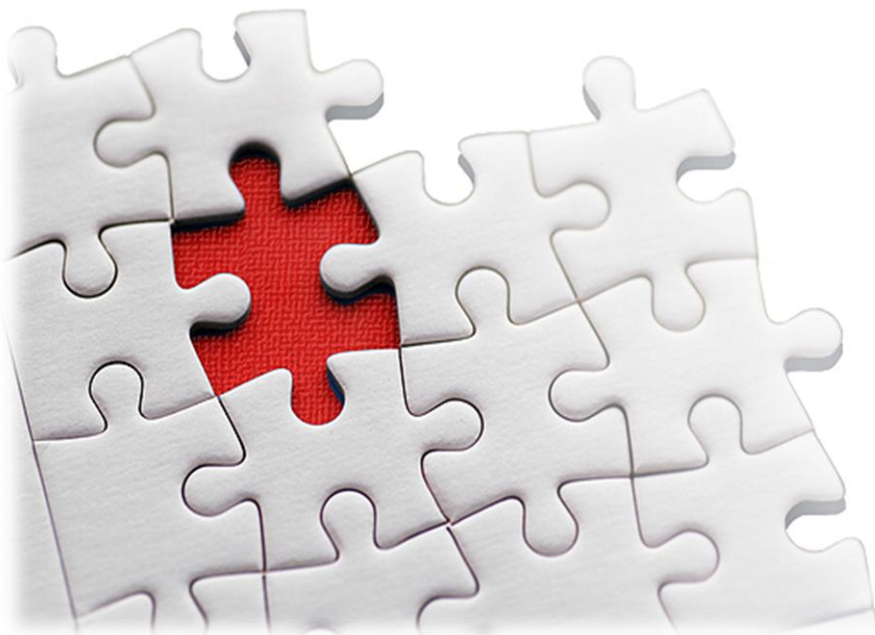
Technology	What technologies may/are used in my applications? Languages, components, services...
Architecture	How does the application structure look like? What is the application architecture look like?
Usage profile	Who uses what? How many times does a end user use per unit time? At what rate do they use a feature/requirement? Are there different modes of usage (end of day, end of month) and what is the profile of usage in each of these modes? What the volume of data that the application is subjected to?
Behavior conditions	What are the conditions that given a behavior of a requirement/feature? How each condition is met - what data (& value) drives each condition?

Summarizing Landscaping is a core concept in HBT (Hypothesis Based Testing) that identifies the various information elements and the process of understanding the element details and their connections enables questions to arise. These questions when answered allow one to understand the application (system), customer and the context, and construct a clear baseline for subsequent stages of testing.

We have also seen that when applying this, we uncover the missing parts of the puzzle and this has helped us to improve/fix the requirements. In a nutshell, good questions aid in early defect detection/prevention and we have used this to test requirements and not only code.

Have a great day!

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



Biography

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

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

He can be reached at ash@stagsoftware.com.



Back To Index





PractiTest

PART 3

Tea-time with Testers Rating: ★★★★★

**by Juhi Verma &
Sharmistha Priyadarshini**

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

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

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

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

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

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

*In this last part, we will continue from the features that we have covered in **May-2011** Issue of **Tea-time with Testers**.*

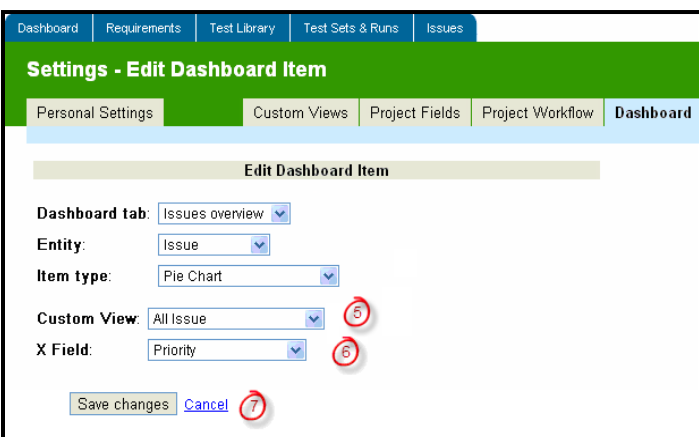
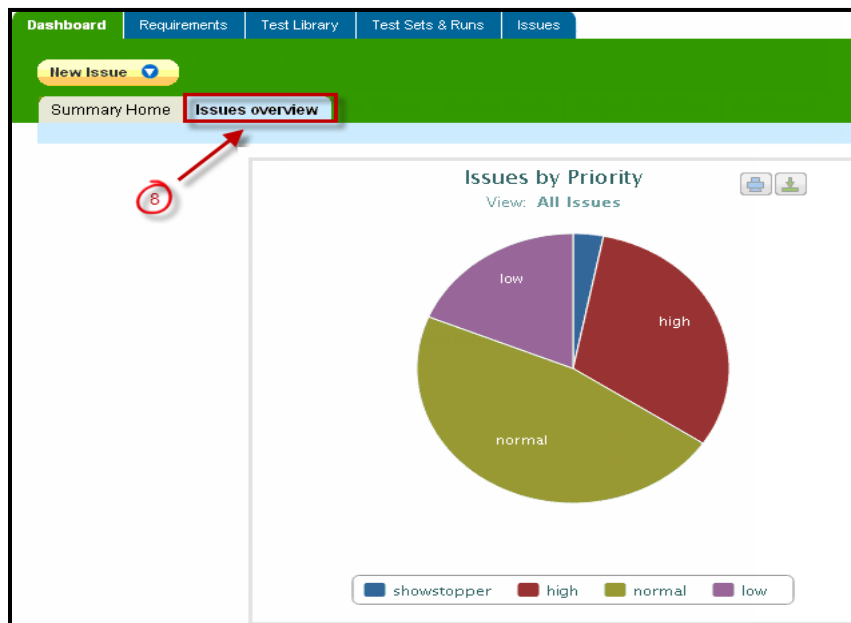
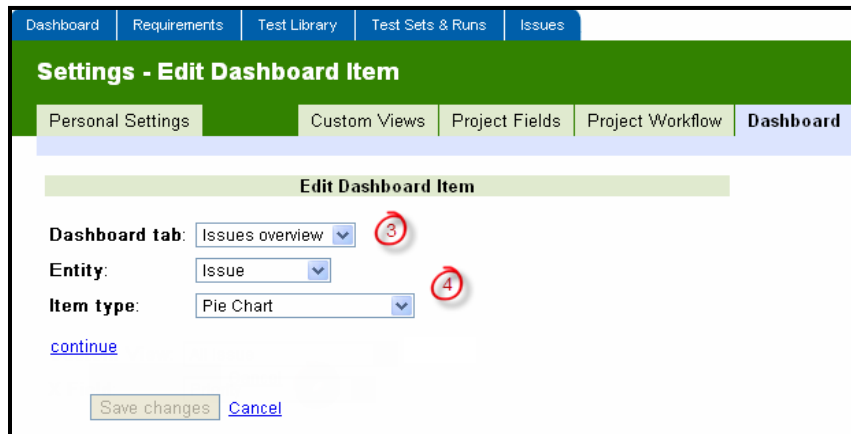
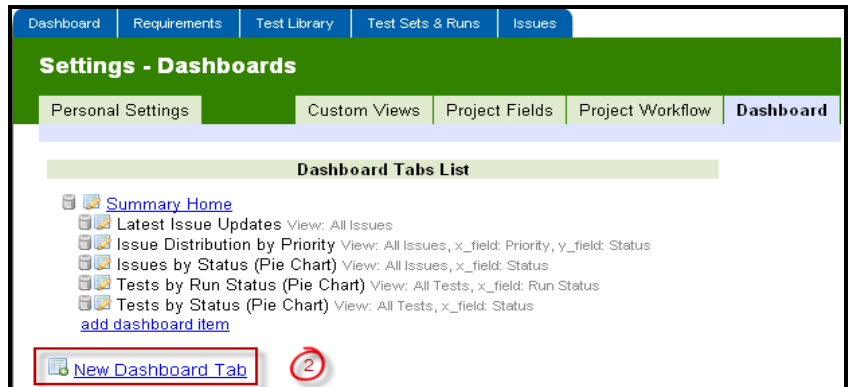
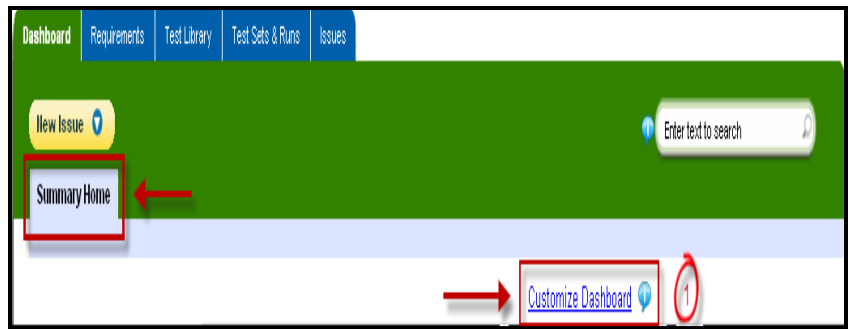
Dashboard

The Dashboard provides overall information about the current state of the requirements, issues, tests and test sets. PractiTest's allows customizing dashboard according to one's priorities & process. You can choose the information to display as part of you dashboard, and even organize the information under different tabs.

You can add new dashboard items under the "Summary Home" tab, or create a new tab with new dashboard items.

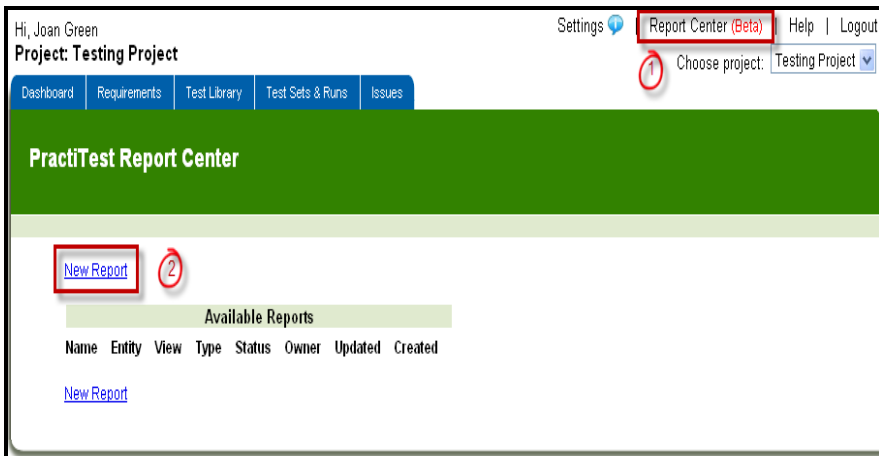
To add a new dashboard tab:

1. Click on the "Customize Dashboard" link on the right side of your screen.
2. Click on the "New Dashboard Tab" link.
3. Select a name for your new tab. For example, "Issues overview"
4. Start adding dashboard items you would like to see. For example, if you would like to see a pie chart of issues, choose the values "Issue" from the Entity dropdown list, "Pie Chart" from the Item Type dropdown list. Click on the "continue" link. Note that each dashboard tab can only hold up to 6 items.
5. Select the issues you'd like to see by selecting a view from the Custom View dropdown list.
6. Select the parameter that will be used for sorting the issues. In this example we have chosen to show all issues, according to priority.
7. Click the "Save changes" button.
8. You can go back to the Dashboard to see your new item under the sub-tab you created.



Report center

The report center allows creating reports and seeing all the previous reports generated for your project. You can create reports based on 3 templates: Tabular, Traceability and Detailed report; and you can use these templates for each of the PractiTest entities (Issues, Tests, Test Sets & Runs, and Requirements).



To create a report:

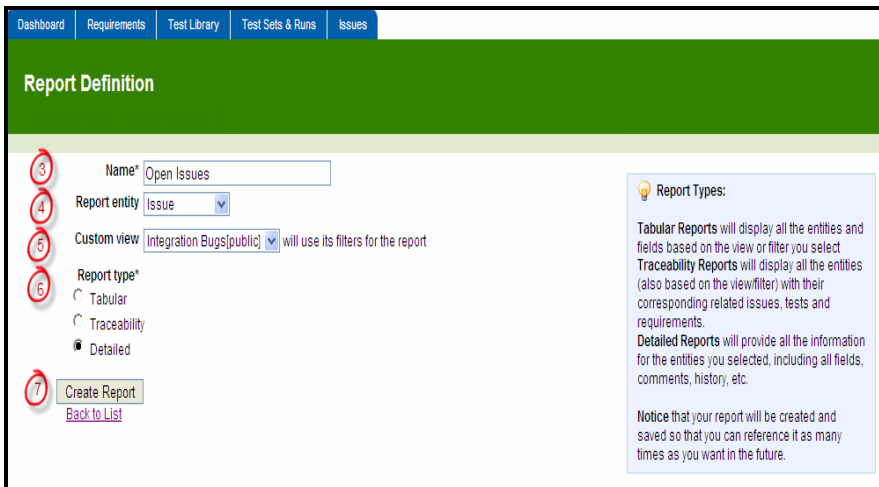
1. Click on the "Report Center" link at the top right side your screen.
2. Click the "New Report" link
3. Choose a name for your report
4. Choose a report entity (for example: issues).
5. Select a view from the Custom View dropdown list.
6. Select the report type:

Tabular Reports will display all the entities and fields based on the view or filter you select.

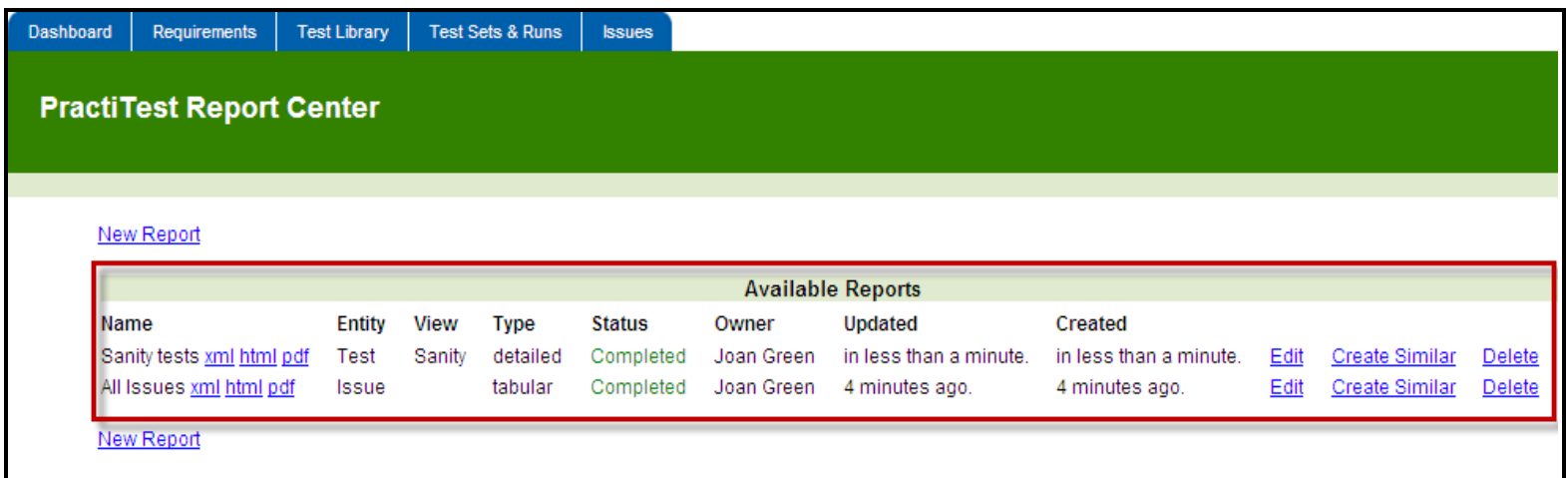
Traceability Reports will display all the entities (also based on the view/filter) with their corresponding related issues, tests and requirements.

Detailed Reports will provide all the information for the entities you selected, including all fields, comments, history, etc.

7. Click the "Create Report" button. Your



Note: your report will be created and saved under "Available Reports", so that you can view it again in the future.



- **Linking of issues with tests, requirements and other issues**

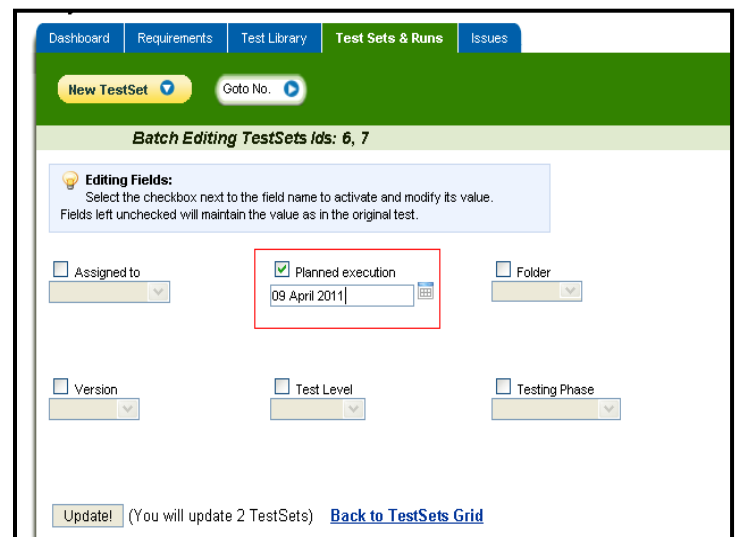
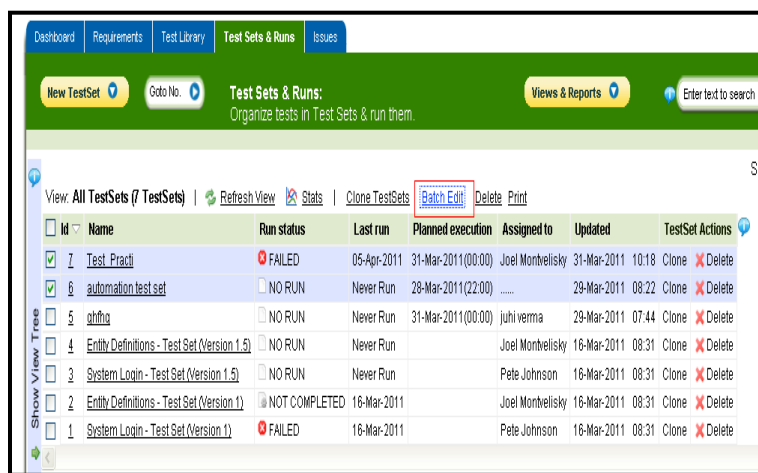
Issues can be linked with other issues; this way you can keep track of your application and project's status, based on the issues raised and their resolution that are linked to your issue.

- **Batch edit**

You can apply changes to Tests, Test Sets & Runs, and Requirements tabs individually based on your project requirements.

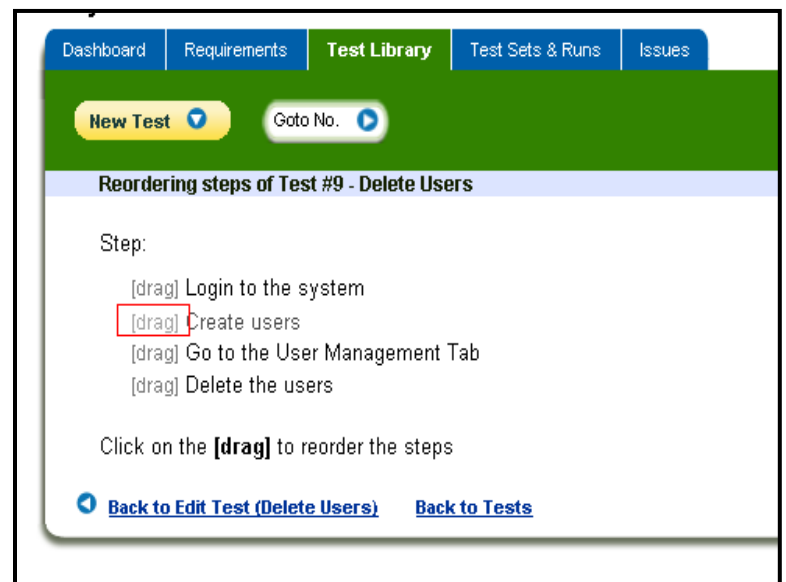
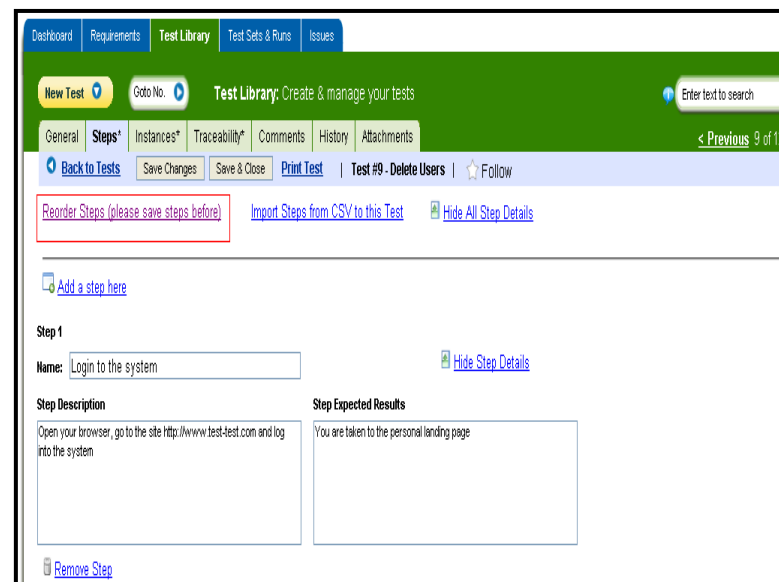
For example, changing the execution date for test sets and runs :

1. Click on the Batch Edit link after selecting the
2. Click on "Update" after making the necessary require changes. Test sets in the Test sets & Runs tab.

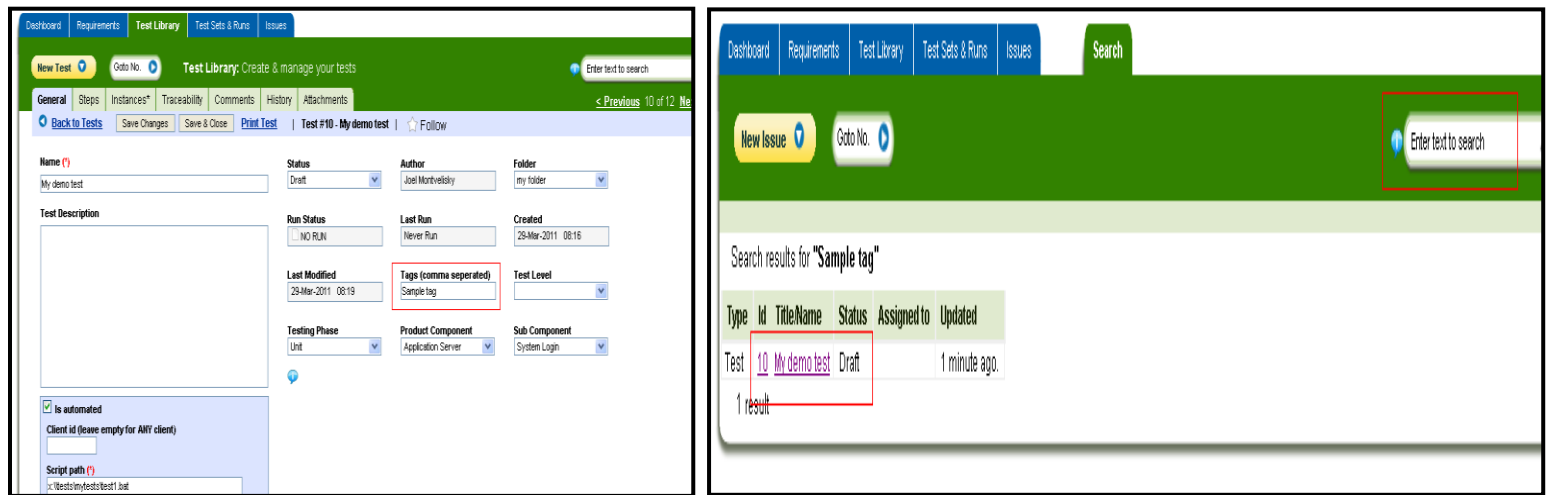


- **Reordering of test steps**

In the Test Library, you can reorder the steps of the already created test case by clicking on the 'Reorder Steps' and dragging the test steps in the opened window.



You can provide tags in requirements, test library, test sets & runs, and issues. For example, tag in test library as shown in the below screenshot:



Can use the Tags for searching in which results are obtained from all the requirements, test library, test sets & runs, and issues with the same tag.

- Using the XBot, you can run [automated tests](#) and scripts created with virtually any tool ([Selenium](#), [Watir](#), [QTP](#), etc) or any homegrown testing framework.

Name (*)
watir login scenarios

Test Description
This will check the login with valid username and passwords, invalid, forgot my password, etc.

Status
Ready

Run Status
PASSED

Last Modified
29-Aug-2010 16:5

Testing Phase
Deployment

☒ **Is automated**

Client id (leave empty for ANY client)
1

Script path (*)
ruby /home/chris/projects/automation/xbot/login_watir.rb

Results path (directory of file)
/home/chris/projects/automation/xbot/output/

Number of result files to upload
2

** PractiTest will be releasing an API that will allow you to create your own custom integrations pretty soon.

Biography



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

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

Juhi also works as a Team member at Tea-time with Testers. She can be contacted at her personal mail id juhi_verma1@yahoo.co.in or on Twitter @Juhi_Verma .



Biography



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

Sharmistha is die hard lover of Data Base testing as she finds it challenging. Being a programmer in past she now loves software testing too as it gives her more scope for analysis. Sharmistha can be reached via her mail id sharmistha.priyadarshini@tcs.com

Do you think that even your own tool should be part of this unique section?*

Feel free to write us. Let the world know what your Tool can do !

To know more write to us at teatimewithtesters@gmail.com

* Conditions Apply

Testing

PUZZLES

by Sebi

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

Send us an answer for the Puzzle
bellow b4 10th July 2011 & grab your

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

Gear up guys.....

Time To Tease your Testing Bone !

Puzzle “Fill the Dots”

Fill the red circles with 12 consecutive numbers in order to match the math expressions shown in picture Bellow:

●	+	●	+	●	-	●	=23
x		-		/		+	
●	-	●	+	●	x	●	=195
-		+		x		x	
●	-	●	+	●	+	●	=28
11		11		11		11	
161		18		0		301	

Answer for Last Month’s Puzzle:

There is a math expression with 10 consecutive prime numbers.

$11+13+17+19*23+29+31+37*41-43=2012$
 $13*17-19+23-29-31+37+41*43+47=2012$
 $17+19-23*29+31+37+41+43+47*53=2012$
 $19*23+29+31+37*41-43+47+53-59=2012$
 $23*29+31*37+41+43+47-53+59+61=2012$
 $29-31*37+41+43+47+53*59-61-67=2012$
 $31-37*41+43+47-53+59*61-67-71=2012$

It starts from the 5th consecutive number, which is 11.

Biography



Blindu Eusebiu (a.k.a. Sebi) is a tester for more than 5 years. He is currently hosting European Weekend Testing.

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

He tweets as @testalways.

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



Every Tester

who reads Tea-time with Testers,

**Recommends it to friends and
colleagues .**

What About You ?

Our Testimonials

Hi Guys,

I am Haripriya, a Software Tester working in Bangalore. I have gone through the Edition of May 2011. It was Awesome. And until now I didn't know these kind of Magazines will be there also. Presently I have a Free time to go on Browse some new Topics. So I asked my Friend to give some site where it can help me. She gave Jonathan Kohl's Web site and there in Publications I found ur Magazine. I just want to be a Smart tester where I can find the Bugs very easily and where I can deliver the Quality Product to the Customer. So please give me some tips to improve My Self.

Regards,
Haripriya,

Dear Haripriya,

Thanks for counting on us. I would recommend that you read all Issues of Tea-time. Any article that we publish will definitely help you to improve your skills.

-Editor

Tea-time with Testers really makes a person think on the kind of quality that can be delivered & Make you feel proud for being a Tester.

It has always motivated me whenever I felt low.

Keep up the Good work Tea-time with Testers!!!

- Mary Oommen, Bangalore

I am very much interested to subscribe to TTWT. It is very much helpful to keep us up-to-date of recent happenings in testing & much more....."

-Kusuma Adayapalam

This is treat for testers... a special magazine which will help provide new changes coming into testing domain and helpful knowledge sharing with all testers.

-

Aniruddha Malvi

"I've heard your magazine is "surprisingly good." I like surprises, so please subscribe me ."

- Curtis Stuehrenberg

"Great Mazaines... enjoyed reading them all... keep up the great work...!"

-

- Ritesh Gudhka

"I have been looking for this kind of magazine. Great Job! Keep doing!"

-Senthilnathan

"Hi I read your may 2011 Issue & found it brilliant for me to study as my per interest."

-Naveddp Tura

"I find this platform very interesting to share the knowledge and experience. I would like to participate in this."

-Avinash Verma

"For the first time I found something like this in the field of Testing! Great!"

-Kumar Prateek

"I would like to enjoy your magazine in the future and will not miss any issues."

-Vladimir Zak

"Interesting..."

-Chris Saunders

"Brilliant. Keep up the good work. Thanks"

-Jan Syrinek

"Thanks for this cup of Teeeee....."

-Kishore Kar

"I read TTWT for the first time and liked it very much. Superb work. Testers Rock..!"


- Rahul Srivastava

"Thanks guys, read one issue so far - really nice and helpful! Excellent job!"

- Andrey

You ask...

We'll help...!



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

- Editor

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

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

in ne>xt issue

articles by -

A photograph of a metal tag with the text "IT'S ALWAYS TEA-TIME" engraved on it. The tag is attached to a chain and a small metal clock. The background is a dark, textured surface.

IT'S
ALWAYS
TEA-TIME

Jerry Weinberg

T Ashok

Joel Montvelisky

Karen Johnson

Gojko Adzik

Michael Larsen

Anuj Magazine

our family

Founder & Editor:

Lalitkumar Bhamare (Mumbai, India)

Pratikkumar Patel (Mumbai, India)



Lalitkumar



Pratikkumar

Editorial | Magazine Design | Logo Design | Web Design:

Lalitkumar Bhamare

Cover Page –Randi Scott Photography

Core Team:

Kavitha Deepak (Bristol, United Kingdom)

Debjani Roy (Didcot, United Kingdom)

Anurag Khode (Nagpur, India)



Anurag



Kavitha



Debjani

Mascot Design & Online Collaboration:

Juhi Verma (Mumbai, India)

Romil Gupta (Pune, India)



Juhi



Romil

Tech -Team:

Subhodip Biswas (Mumbai, India)

Chris Philip (Mumbai, India)

Gautam Das (Mumbai, India)



Subhodip



Chris



Gautam

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

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

Google™

Join our community on

facebook.

Follow us on



www.teatimewithtesters.com

Give Feedback

