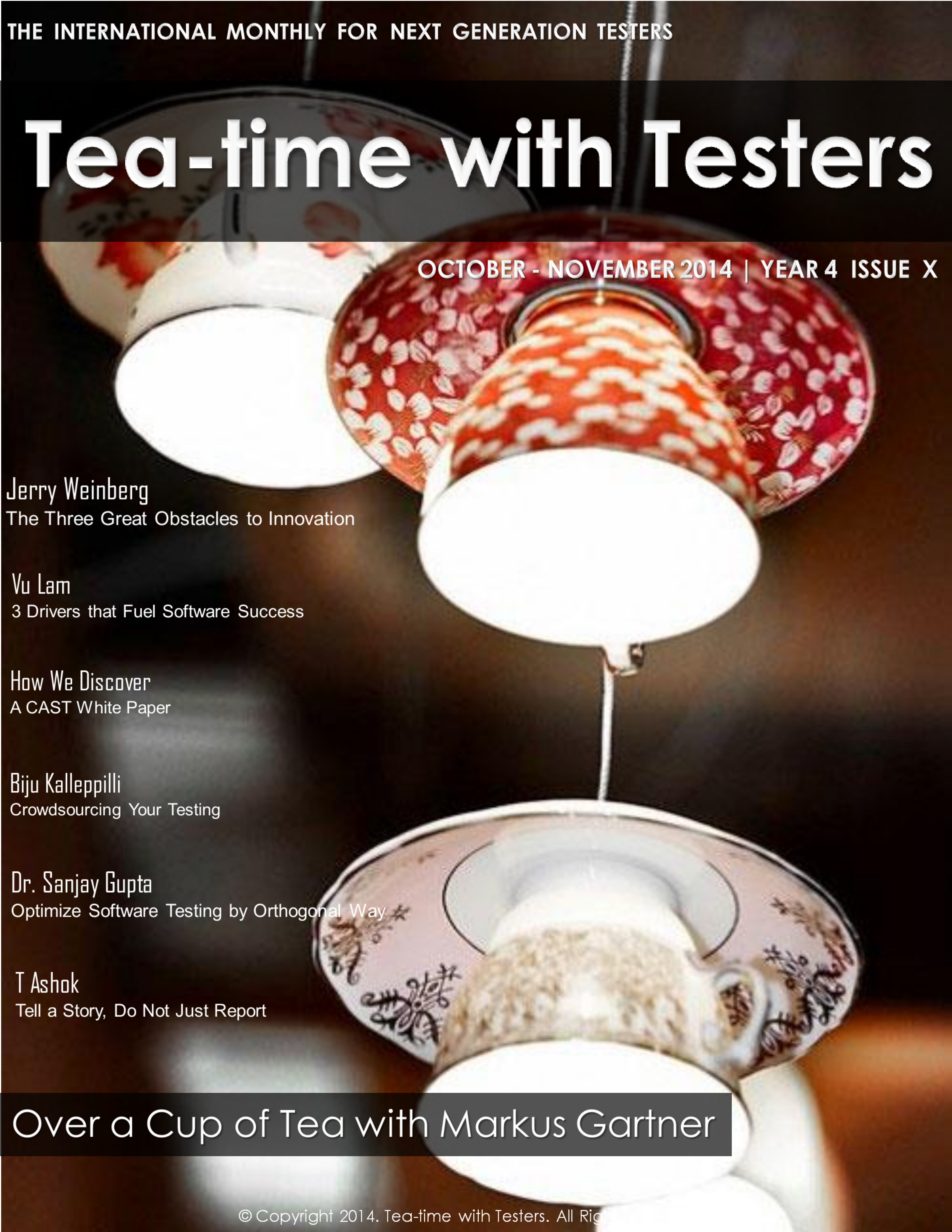


THE INTERNATIONAL MONTHLY FOR NEXT GENERATION TESTERS

Tea-time with Testers

OCTOBER - NOVEMBER 2014 | YEAR 4 ISSUE X

testomaton

Quality Assurance via Automation

Software testing startup specializing in test automation services, consulting & training for QA teams on how to build and evolve automation testing suites.

SERVICES

Test System Analysis and work estimation

Review of client's system (either in development or on early stage) to produce a Test Plan document with needed test cases and scenarios for automation testing.

Tests creation and Automation

Development of test cases (in a test plan) and Test Scripts to execute the test cases.

Regression and Test evolution

Development of Regression test suites and New Features suites, including test plans and test scripts or maintenance of existing.

Architecture Consulting

Review of software architecture, components and integration with other systems (if applicable).

Trainings

Depending on the particular need, we can provide training services for: Quality Assurance theory and best practices, Java, Spring, Continuous Integration, Groovy, Selenium and frameworks for QA. For further information please check our web site.

We enjoy putting our experience to your service. Our know-how allows us to provide consultation services for projects at any stage, from small up to super-large. Due to our range of expertise we can assist in software architecture and COTS selection; review of software designs; creation of testing suites and test plans with focus on automation; help building quality assurance teams via our extensive training curriculum.

look us up: www.testomaton.com - twitter: @testomaton

Get ready for your gift… this festival season!

IN THROUGH THE SIDE DOOR

A video talk by Michael Larsen

Stay tuned for more updates…

WWW.TVFORTESTERS.COM

TEA-TIME WITH TESTERS

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

Created and Published by:

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

Editorial and Advertising Enquiries:

Email: editor@teatimewithtesters.com

Pratik: (+91) 9819013139
Lalit: (+91) 8275562299

Editorial

When delays are for Good…

First of all, I want to thank all of those who wrote us letters enquiring about coming issue. It gives us immense pleasure to learn that even after almost 4 years, our readers are as passionate about us as they were in our initial days.

With the launch of first issue of Software Tools Magazine, we had decided to give small break to our readers so that they can digest next issue of Tea-time with Testers with ease. But I was wrong! Dozens of emails eagerly enquiring about next issue of TTwT has made us realize the real appetite of our readers and I promise that we will continue to feed you with same passion in future.

There are bunch of other things we are working on to give you a new year treat and I am sure that our readers will appreciate those special efforts.

On that note my friends, allow me to take your leave for now. To keep my promise about your New Year gift, I must rush!

Until then….

Bhamare

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

STC 2014

14th Annual International Software Testing Conference

www.qaistc.com

14th Annual

International Software Testing Conference (STC 2014)

December 04 - 05, 2014 | Bangalore

Re-invent Testing – Inspiring Innovation & Increasing Agility

sharing & exchange of experiences IDEAS & **LEARNING**

800	131	050	005	004
PRACTITIONERS	ORGANIZATIONS	SPEAKERS	AWARDS	TRACKS

The 14th edition of STC 2014 will take place on 4-5 December 2014 in Bangalore, India. Built on the theme of Re-invent Testing – Inspiring Innovation & Increasing Agility, the conference aims to be a display of ideas, experiments and experiences to explore challenges and suggest techniques and Best Practices to successfully create inspired innovation and increased agility.

MAJOR TRACKS

1. Business Leadership
2. Program Management
3. Tools & Techniques
4. Emerging Areas

ORGANIZED BY: QAI

IN ASSOCIATION WITH: ETI

QAI Delhi (India): +91-11-47776666/30 | QAI Mumbai (India): +91-22-2880-7306/07/08 | QAI Bangalore (India): +91-80-49023452/53/56

QAI Singapore: +65-6225-8139 | QAI China: +86-21-51314155 | QAI USA:+1-407-363-1111

QuickLook

STM
NOVEMBER - 2014
YEAR I ~ ISSUE I
SOFTWARE TOOLS MAGAZINE
© Copyright 2014. Tea-time with Testers. All Rights Reserved.

AGILE designer

AGILE DESIGNER

FREE TRIAL

QASymphony

SmartBear SOFTWARE

QASymphony Announces Integration with Rally Software

ATLANTA, GA – November 18, 2014 —QASymphony, leading provider of Agile testing solutions, today announced a partnership with Rally Software (NYSE: RALY), a leading global provider of enterprise-class software and services solutions to drive business agility. QASymphony's qTest test case management tool now fully integrates with Rally's Agile Platform for Application Lifecycle Management (ALM).

This enhanced qTest integration to Rally® ALM allows customers to leverage the power of exploratory, manual, and automated testing all in one easy to use platform. Additionally, customers are granted seamless traceability out of one tool and in to the other. qTest is a scalable QA software testing management platform optimized for enterprise Agile development teams. Available as a SaaS application or installed on-premise, qTest allows teams to manage requirements, design test cases, plan test execution, track defects, and generate status and quality-metrics reports.

"We are thrilled to work with Rally to provide a best-in-class testing solution with the most intuitive integration possible for Rally's customers," said David Keil, CEO, QASymphony. "Following a successful joint customer implementation, we are proud to now introduce this integration to the Rally user base as a whole, and we'll continue to solicit customer feedback and input to ensure that our customers create better software. QASymphony is the testing platform of choice for enterprise customers and our partnership with Rally is a strong component of our solution."

"Rally is committed to helping our customers produce higher-quality software that best meets end user expectations by successfully leveraging Agile to meet business challenges," said Ryan Martens, Rally founder and CTO. "Our partnership with QASymphony is designed to bridge the gap between developers and testers so that developers can spend less time with documentation and more time testing and advancing their products and services to market quickly."

Rally customers span the Fortune 500, government, financial, healthcare, media and entertainment, technology, and telecommunications industries. A list of Rally customers can be found here: https://www.rallydev.com/all-rally-customers.

About Rally

Rally Software provides leading software and services solutions that drive agility. Companies work with us to accelerate innovation, improve performance, and respond to evolving customer needs. Rally's SaaS platform transforms the way organizations manage the software development lifecycle by aligning software development and strategic business objectives, facilitating collaboration, and increasing transparency. Rally's consulting and training services apply Agile and Lean approaches to help organizations innovate, lead, adapt, and deliver. http://www.rallydev.com.

Rally, Rally Software and the Rally logo are trademarks or registered trademarks of Rally Software Development Corp. in the United States and other countries. All other trademarks are properties of their respective owners.

About QASymphony

QASymphony's test management and agile testing solutions help teams create better software. With QASymphony's tools businesses can accelerate testing to keep up with the pace of today's development to ensure that productivity gains from agile development are matched with the oversight, visibility and control required to build quality software. Empowering quality software at companies such as Adobe, Barclays, BetterCloud, ExactTarget (Salesforce Marketing Cloud) and Vonage, QASymphony enables teams to communicate and collaborate faster, bringing visibility and control back to the development and testing lifecycle. The company is headquartered in Atlanta, GA.

Website: www.qasymphony.com
Facebook: www.facebook.com/qasymphony
Twitter: www.twitter.com/qasymphony

# # #

Press Contact:

Victor Cruz

Principal, MediaPR

vcruz@mediapr.net

Quality Testing reaches 15000+ registered members

Mumbai, India – 30 November 2014 - **Quality Testing** (http://www.qualitytesting.info), a leading online-social network for software testers has crossed a big milestone of reaching 15,000+ registered members.

With 15,010 registered users, Quality Testing has now become one of the few large online communities for software test professionals. With such a huge user-base, Quality Testing has also become largest testing community of Asia.

Quality Testing's motive is to benefit the testing community by providing an open platform for knowledge sharing and networking between testing professionals worldwide. It has attracted many enthusiastic test professionals from 200+ countries with major registrations done from India, USA, UK and Canada.

More than 4350 professionals visit Quality Testing every day. This community is currently accessible to members as well as non-members. Quality Testing considers its user base and forums as its valuable assets.

Quality Testing also has a news portal "Latest Software Testing News" and runs a jobs board "Quality Jobs Portal" in association with Tea-time with Testers. That's not just it. You will find content services, hundreds of technical videos, discussions, featured interviews, details of software testing events from across the world, a weekly news-letter, blogs, and specially created groups on various subject areas in software testing in this terrific community.

Speaking on the occasion, Mr. Kiran Kumar (Founder of Quality Testing) said, "We are happy that we have reached another milestone. This is largest non-commercial and non-profit social network for software testing professionals across the world. Reaching this milestone has made us more responsible and we are committed towards making Quality Testing a most definitive platform for social networking between testing professionals. We would like to thank everyone who has registered on QT and making use of our forum. It wasn't not possible without them."

Tea-time with Testers is strategic partner of Quality Testing and we congratulate them for this candid achievement.

Back To Index

Contact: contact@qualitytesting.info

Latest Software Testing News

- Know about the Industry

www.latestsoftwaretestingnews.com

A million dollar smile ?

Ask our sales team about
our **Smiling Customer** ☺ programme

*Adverts starting from $100 USD | *Conditions Apply

Photo credits: Amir Eltahan

Markus's stand for quality, his work in Agile and ATDD circles is known by all. He is a software craftsman who specializes in testing, development, project management, people and general systems thinking. He is a black-belt tester and instructor in the Miagi-Do school of software testing, and a co-founder of the European chapter of Weekend Testing.

I got to meet Markus at CAST 2014 Conference and meeting him in person was a great experience. From his testing career to his stand about testing, we discussed about many different things. And I am glad that we talked.

Read on to find out what all we talked about, in this exclusive interview.

- Lalitkumar Bhamare

Over a Cup of Tea with Markus Gartner

Please tell us about your testing journey.

In 2006 fresh from university, when I was looking for a job, I applied for a position as a release manager. In the first few minutes in the job interview, I was told that they no longer had the position I originally applied for open, and that I was interviewing as a QA Engineer. I thought “ok”, and got the job.

Having had actually no exposure to software testing during my time at the university, I needed to learn what it takes to be a good tester. Eventually I found out that I had been a tester all my life already.

One and a half years later I found myself in the role of a test group leader. I had four colleagues in my group, and we were faced with the situation that we had to maintain an automated testing code base – that we originally implemented – that was hard to maintain with 12 people – and we had to deal with just us five.

So, I figured we need to do something different, and that’s how I discovered Agile methodologies. Inspired by those materials, we jumped in, and replaced what had “grown” for a year by a new approach while the project with the customer was still going on. 18 weeks later we had a faster test automation suite that was able to answer the crucial questions from project managers and developers.

A couple of years later I moved on as a trainer, coach, and consultant for agile software development. That’s what I am still sticking with, but my heart still lies with the testers.

Do you have any regrets being a tester by chance? Would you have loved to be a tester by choice?

Well, it was my choice to go for the software tester job I was interviewed for. I could have moved on, and maybe pursue a Ph.D. program – as I originally wanted to do. But then, that job was way too much fun to move on.

It was a chance that opened that day, and I made the choice to see what it takes to become a great tester. I didn’t pursue that career during my time at the university, and I probably wouldn’t have pursued it. Programming was way more fun back then. On the other hand, in test automation I found I can combine both, my programming skills, and my attitude to be nitpicky. That’s what made me stay, and that’s why I don’t regret it.

You are well known in industry for your contribution towards ATDD/TDD/BDD. Please tell us more about your work. How did it start? What made you get involved more in it?

When we faced the problem to replace that test automation solution based on shell scripts, I was worried about how testers on agile projects could keep up the pace with development. So, I not only dived deeper into agile testing, but also agile software development, and context-driven testing. As I said we eventually replaced the old solution within 18 weeks by applying lots of lessons I could learn from the early books on ATDD and TDD.

I remember that I kept on coming home from work with all those questions in mind. When I came back the next day, I had read a bit more about the approaches that others took, and was inspired enough to immediately try it out.

I think my knowledge was mostly driven by the need to learn more, and try it out close to immediately. I experimented a lot with all the stuff once we had freed our back. For example, I remember one year between Christmas and New Year’s. I read Growing Object-Oriented Software Guided by Tests back then. We had recently gone through a larger project, and didn’t have unit tests for our test automation code in place. I knew that was a risk for the long-term support of that project. So, I dived in, and used lots of mocking to bring out test automation code base under test. That was a lot of fun, and I learned tremendously in those 5-7 working days about mocking and unit tests.

Do you think Test Driven Development or Behavior Driven Development are the best ways to develop software?

I believe they are among the best that we currently know. Rumor has it that "something like TDD" was already used by folks like Jerry Weinberg, Bernie Dimsdale, and John von Neumann.

What? Really, here's a passage from an interview between Michael Bolton and Jerry from 2008 (http://www.developsense.com/blog/2011/01/jerry-weinberg-interview-from-2008/)

... CONTINUED ON PAGE 47

The Three Great Obstacles to Innovation

Technical stars are innovators—that's how they got to be stars—but their skills as individual innovators often obstruct their climb to a leadership plateau. When they're not aware of their own innovative processes, they may lose their powers when they assume leadership responsibilities. If they don't understand the sources of innovation, their own attempts to innovate may destroy the creative environment for others

In the following chapters, we'll explore the major obstacles to innovation and how they can be overcome, both in yourself and in others.

Even in her childhood she extracted from life double enjoyment that comes usually only to the creative mind. "Now I am doing this. Now I am doing that," she told herself while she was doing it. Looking on while she participated. —Edna Ferber*, So Big*

Enough of theory. What are you actually doing that either makes you creative or blocks your brain? Can a new model of leadership really work for you, or is it only as useful as the latest diet fad? The theory of dieting is simple enough: All you have to do is achieve a balance between input and output. According to the balance theory, I shouldn't have this bulge around the middle, but look at me.

Theories are like that. I can give you all sorts of techniques for developing your problem-solving or leadership abilities, yet find that you're making no progress. In this chapter, I'd like to explore the most common obstacles to progress toward problem-solving leadership, starting with the question of why I sometimes put on pounds without knowing why.

ARE YOU AWARE OF WHAT YOU HAD FOR DESSERT?

Keeping trim is important to my business, because people pay more attention to what you do than what you say. As a consultant who's supposed to help streamline organizations, I make a poor model if I'm a not exactly streamlined myself. At one such organization, Dani and I were looking for the source of their diminished productivity. After we spent the day observing and interviewing, Shirley, the manager of the systems analysis sections, invited us home for dinner.

Shirley and her husband, Harrison, lived with their three sons in a busy but attractive neighborhood, which seemed well matched to Shirley's temperament. When I raided Shirley's refrigerator for a pre-dinner snack, I was surprised to find evidence that she was struggling to control her weight. There were low-cal foods inside, calorie charts on the door, and a box of appetite suppressants on top. As she prepared supper, we discussed our common problem.

"I must have a metabolic problem," I said.

"Me, too. I used to have trouble with snacks, but I've controlled that, yet I still can't lose weight."

"Maybe it's eating out. You know, Dani and Harrison have already promised the kids we'll go out to Swenson's after dinner. I'd rather stay home because I can't resist their ice cream."

"Well, I can resist," Shirley boasted. "I'll just have a cup of coffee."

"Maybe if I eat a full supper," I said, "I won't have any appetite for dessert." But there wasn't much conviction in my voice.

I ate a full supper, but at Swenson's I wasn't able to resist ordering a dish of raspberry sherbet. Dani had a small marshmallow sundae, Harrison had a large banana split, the two oldest boys had special children's sundaes, and the youngest had a chocolate chip cone. Only Shirley resisted, righteously and vocally ordering a cup of coffee. I noticed, though, that Shirley took cream in her coffee. And sugar.

Then when Buddy's cone arrived, Shirley seemed quite concerned that it was going to fall or drip. "I'll just trim off the extra," she announced, and proceeded to reduce Buddy's scoop to half its original size. Somehow, the excess went into her mouth.

In the meantime, Harrison offered her a "taste" of his banana split. "Just a taste," she said, but he insisted that she taste each of the three delicious flavors, plus each of the three syrups, some of the whipped cream, a bit of the banana, and both cherries. Before long, the dish was resting halfway between them, and she was using her coffee spoon to continue her "tasting."

She also sampled the other two children's sundaes, nobly eating the parts they didn't like. To wash all this down, she ordered a refill on coffee, then another, both with cream and sugar. Somehow, there seemed to be a food magnet in her mouth, irresistibly attracting all loose food on the table. I found myself compelled to offer her some raspberry sherbet, but she was busy finishing one of the decorative cookies.

Later that evening, with the kids tucked into bed, we four adults sat around having a nice chat. Around eleven, Shirley asked if anyone was hungry. None of us were, but she slipped into the kitchen anyway. When she returned with a plate of cheese, fruit, and nuts, she announced to nobody in particular, "Well, you guys all had ice cream, and I didn't have any."

"But you did," I observed, regretting it immediately.

Shirley regarded me with a puzzled look. "No. Don't you remember? I didn't order any ice cream. Just coffee." True, she hadn't *ordered* any ice cream, but didn't she realize that she'd eaten at least twice as much as anyone else at the table?

SELF-BLINDNESS: THE NUMBER ONE OBSTACLE

For many people, work is like Shirley's eating. They waste time pursuing dead ends, dragging out phone calls, or getting involved in useless arguments, yet never realize why they don't accomplish anything. Writers, for example, throw away many false starts, or sit staring at a page for hours without typing a single word. Many programmers, stuck in some fallacious argument, search for a single error long past the point where they should seek help.

I know that other people do those things (Dani and I watch them all the time), but I'm sure that I don't do them—not very often, anyway. Wouldn't I remember if I did?

In fact, I wouldn't remember, any more than Shirley could remember what she had for dessert. Like everyone else, I'm quite unable to see myself in action, particularly when I'm exhibiting my least productive behaviors.

Whenever I watch someone like Shirley not watching herself, my consciousness is raised, for a few days anyway. I snack less and lose weight. But soon I stop watching again, and my spare tire starts to in ate. What I need is consultant, someone to watch me eat and report to me what I can't see for myself. The same is true for all of us. The only way we can see ourselves is through other people.

This inability to see ourselves as others see us is the number one obstacle to self-improvement. The great majority of would-be problem-solving leaders are stuck on this one obstacle. To surmount it, they must recruit others to help them. Probably the best way to get someone to watch you is by making a pact to watch that person in return. Even a mutual observing relationship is a rather delicate one, so it may take some time to develop a relationship that works well.

Whatever you do, keep it mutual. Don't ever volunteer your observations about people as I did with Shirley, no matter how helpful you think it would be for them. Shirley was nice enough not to punch me, but I was lucky.

Even when people ask for your observations, they won't always like what you have to say. I once asked Dani to watch my eating patterns. As an anthropologist, she's an expert observer—so good, in fact, that it almost broke up our marriage. It's just not much fun to be watched all the time, so whatever you do, don't pick your spouse as your watching partner.

To be continued in next issue…

Back To Index

Biography

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

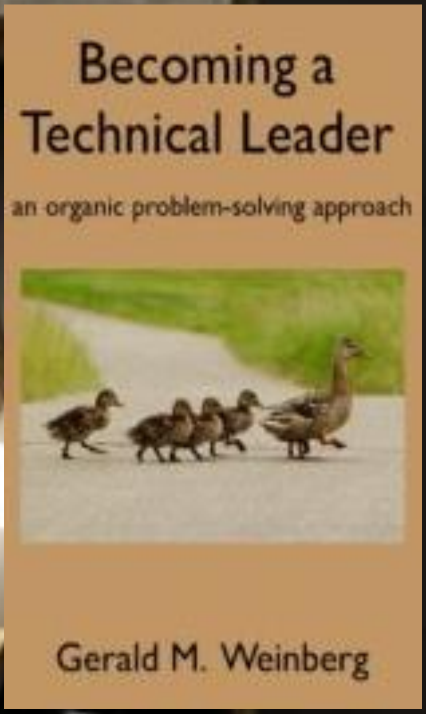
For more than 50 years, he has worked on transforming software organizations. He is author or co-author of many articles and books, including The Psychology of Computer Programming. His books cover all phases of the software life-cycle. They include Exploring Requirements, Rethinking Systems Analysis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

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

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

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

Becoming a Technical Leader is a personalized guide to developing the qualities that make a successful leader. It identifies which leadership skills are most effective in a technical environment and why technical people have characteristic trouble in making the transition to a leadership role. For anyone who is a leader, hopes to be one, or would like to avoid being one.



This is an excellent book for anyone who is a leader, who wants to be a leader, or who thinks only people with 'leader' or 'manager' in their title are leaders.

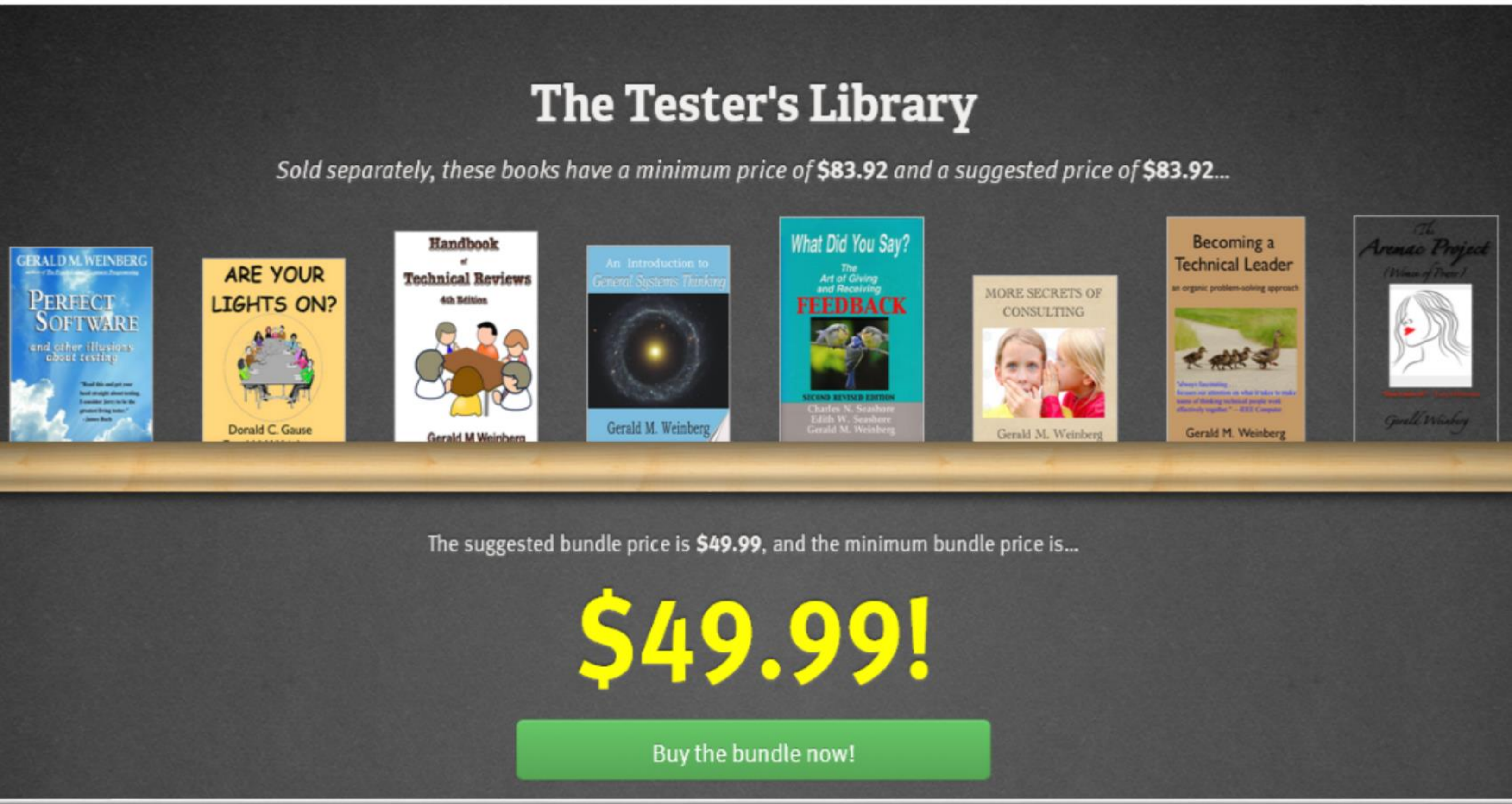
TTWT Rating: ★★★★★

Its sample can be read online.

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

The Bundle of Bliss

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



The Tester's Library
Sold separately, these books have a minimum price of $83.92 and a suggested price of $83.92...
The suggested bundle price is $49.99, and the minimum bundle price is...
$49.99!
Buy the bundle now!

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

The 8 books are as follows:

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

Know more about this bundle

TEA-TIME WITH SMARTBEAR

About this column…

SmartBear Software not only provides testing tools to help development and testing teams accomplish their software quality goals, it is also a hub of information and news for the software testing industry. From workflow methodologies to discussions on industry practices and tech conference coverage, SmartBear has become a source for testers seeking quick access to a wide variety of content.

SmartBear's goal in creating this column in **Tea-Time with Testers** is to empower software testers around the globe by helping them become more informed about the current state of the software testing industry.

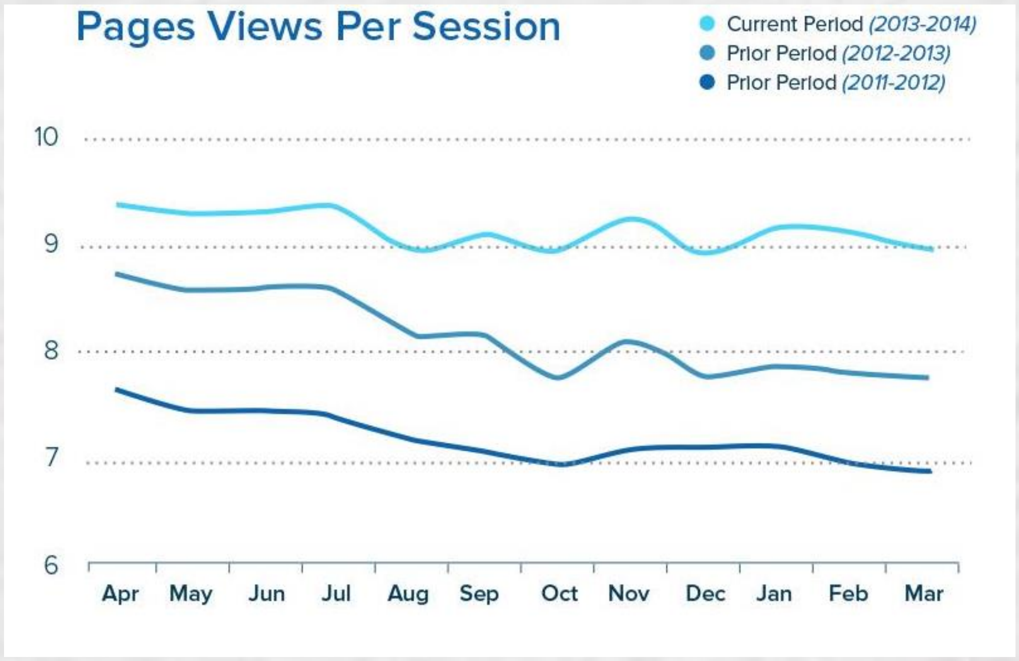
This Holiday Season Help Your Website Achieve Peak Performance – Part 2

- by **Nikhil Kaul**

In part one of this series, I touched upon the enormous opportunity holiday season presents for any internet retailer. While this all sounds like great news for e-retailers, the holiday season comes with its own set of challenges. Even though customer spend and traffic continues to increase on year-over-year basis, not all retailers have been able to use this to their advantage during the holiday season because they were not well prepared. *Toysrus.com* for instance experienced a 14 hours of downtime during the 2013 holiday season. In other words, many of the visitors who were trying to shop at Toysrus.com during those 14 hours, ended up making purchases at one or more of its competitors. Remember, unless you are selling something that is truly one of a kind, your competition is only about 3 clicks away – and I don't know about you, but personally, I'm *far* more likely to invest 3 clicks to go to a different store if I am dissatisfied, if it's taking too long, or if it's closed than I am to drive across town to see if the competitor is will provide service more to my liking.

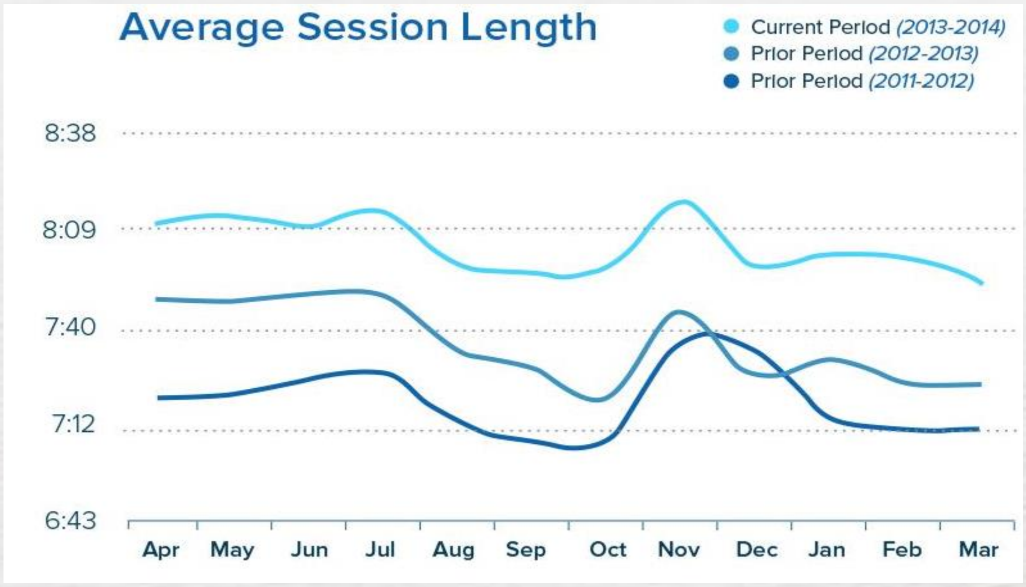
In the case of Toysrus, their moderate page load time of 4.089 sec in comparison to Target's 2.55 sec and BestBuy's 3.494 sec, certainly didn't help matters for them. Toysrus.com definitely ended up losing some of its impatient customers to competitors even when the website was up and running.

On the first look, a one and a half second delay in page load may not appear to make a big difference, but think about it. How many pages do you visit while you're shopping for the perfect gift? How many half seconds is that compared to how many half seconds it takes you to get to competing e-retailers whose responsiveness isn't undermining your holiday spirit while you're gift hunting?



To fully grasp how much difference this half second can make, let's take a look at some trends in how consumer behavior is changing while making purchases online. Research conclusively shows that consumer attention continues to dwindle to new lows every year. Average Session Length and Page Views per Session, both metrics which demonstrate consumer attention, have hit new bottoms for each of the past three years. These dwindling attention spans means if e-retailers don't pay special attention towards site performance, they will, not might, but *will*, end up losing customers to competitors.

Throughout the remainder of this series of blogs, we're going to explore specifically how the much difference a half second can make, how easy it is for the traffic increase of the holiday season to slow your site down by that half second, and specific things you can do to keep your holiday revenues from falling victim to that dreaded half second (or worse, downtime!).



Tea,

Testing and

QASymphony

3 Drivers that Fuel Software Success

When it comes to Software Quality, focus on the end user

There are many different ways to determine how "good" a piece of software is. A developer might feel that efficient code that's maintainable and extensible is the most important thing. A product manager might be looking for a rich feature set that delivers more than the competition. A tester might be aiming for a bug-free release. Problem is, all of them may be wrong: you can still end up with a product that is too complex, too hard to use and too hard to learn.

If you really want to know how good your software is then you should be asking your end users. The real measure of quality is revealed in your usage statistics. What are your adoption rates and how often are people using your product? If people use it a lot, then they like it, and if they don't…well, there's a reason for that too.

The 3 drivers of software success

You need a solid foundation built on three drivers in order to deliver good software and each one must be measured and judged from an end user perspective.

1. Usefulness

Does it have the features and functionality that people really want? It must do what the end user expects it to do. It's not about having more features than the competition; it's about having the features your end users want and delivering them to a high standard.

2. Usability

How easy is it for a person to use? It must be fast and efficient, easy to learn and understand, and capable of recovering graciously from any problem. The learning curve should never be too steep, and there's no patience for long complex user guides.

3. Appeal

What's the reason that the end user should install your software and load it up again and again? The best software is something that people will immediately see the value in, and after using it they'll want to use it again every day, not once or twice a year. Would the user tell their friends and colleagues to try it out?

Understanding the end user

For any software tester the first thought should be to understand the end user. You can focus on the first two pillars and always ask how useful and usable is this software? You'll obviously explore the reliability, the flexibility, and the extensibility of the software using the full range of tools at your disposal, but try to do this with the same mindset as your intended audience.

Practically speaking, this means learning the end user's context. Don't just rely on documentation; you need to immerse yourself in their domain and gain an insight into the specifications they require and the desires they have.

The simplest route to this data is to talk directly to them. Modern projects frequently involve a lot of discussion with the end user and agile development dictates that a feedback loop should be established early and feedback should be collected often.

When the product manager organizes feedback groups and meetings to discuss the expectations and job requirements, make yourself a part of that conversation. The earlier everyone gets involved, the better informed they will be, and the better the job they will do. If you don't get involved until later then make a point of being proactive in catching up. Identify representative end users and interview them.

Going beyond bugs

The old metrics are outdated. Reports on defect counts and defect density simply don't provide a clear picture of software goodness. Just because a piece of software is completely bug-free that doesn't mean that it's good. Testers need to look past defects and consider things like how easy it is to learn to use and how well it gets the job done.

For end users, defects are obviously a problem, but you can't eradicate every defect before release, especially when you're rolling out new features frequently into a live product. If we accept this then we can take a different view. What's more important is how the software handles a problem. Can it recover? Can it provide clear instructions for the user? Is it obvious to them what happened and what they should do next? The help and support should be built-in.

Make the end user central

It's time to change the way that we evaluate software goodness. Instead of collecting a disparate group of metrics and snapshots that deliver insights on specific elements of the development, we should make the end user central and narrow our focus. Even an efficient, feature-rich, defect-free product can be hard to use.

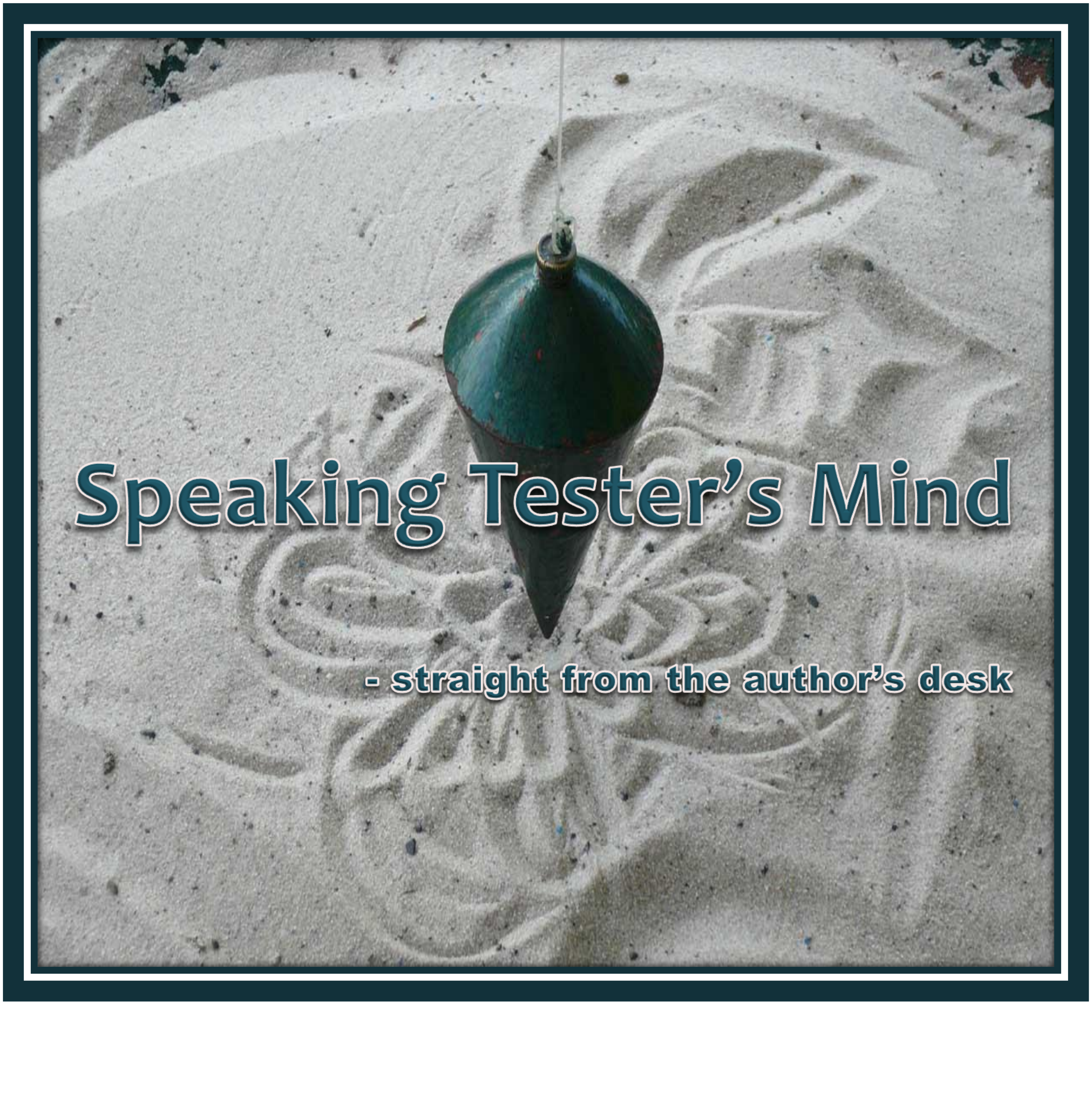
Any start-up or seasoned software team would be wise to consider the three pillars that will drive usage before any requirements are drawn up or a line of code is written, and to continue in that vein for the full life cycle of the product.

Let the end user lead the way. Take the time to understand what they want in a product, so you can make it useful. Observe how they interact with the software, so you can make it usable. Never take your eye off the adoption and usage statistics, because they are the only measure of the product's appeal that really matter.

Vu Lam is founder of QASymphony, a leading provider of test management platforms for agile development teams, He was previously with First Consulting Group and was an early pioneer in Vietnam's offshore IT services industry since 1995. He holds an MS degree in electrical engineering from Purdue University.

You may reach him at vulam@qasymphony.com

Back To Index


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

How We Discover

A CAST White Paper

Abstract

In Exploring Science: The Cognition and Development of Discovery Process [3], David Khlar et al. conducted experiments to analyse the process of scientific discovery. Twenty-six software testers repeated Khlar's experiments using robots to explore how we discover and how it relates to the process of software testing. Participants worked in groups to perform 1 of 6 experiments. All experiments required participants to discover the function of a button on the robot. Experiments differed in the type of instructions provided, and the size of the group. The groups recorded their experiments and outcomes, then drew conclusions about how discovery takes place, the variables that affect discovery, and how that discovery relates to software testing. The results of the study are similar to the findings in Khlar's book; the difference being how these results relate to software testing.

Introduction and Motivation

This paper investigates the way software testers discover. In software testing, one must seek and discover bugs, requirements, and other various information to make better informed decisions on how to test, analyse risk, and evaluate the product. James Bach writes, "Testing is questioning a product in order to evaluate it." [1] Cem Kaner defines testing as an "investigation done to provide stakeholders, information about quality of a product or a service."[2] In both definitions, we can see that testers need to discover.

Then the question we pose is -- what does it to take to discover? In this paper, we break this down and consider whether prior knowledge or any biases have an effect. If so, why? How about the amount of information, amount of direction, or size of the team? Is it better to take a step back or to drive and delve into the problem head first?

This paper uses an experiment performed by Klahr and Dunbar asking participants to determine how a device works and relate this to testing. With the use of 'Big Trak'[1] and 'Big Trak Jr.[2]' robots [4] and unknown functionality, we conduct a series of experiments to investigate how we discover.

Related and Prior Work

Klahr & Dunbar (1988) has already done work with respect to observing discoveries and scientific thinking using a programmable robot, 'Big Trak' [4]. Participants were asked to determine the functionality of a certain button. In this paper, they present a model of Scientific Discovery as Dual Search (SDDS) in which there are two primary spaces of scientific discovery: hypothesis and experiments. The hypothesis space is determined by an experimenter's prior knowledge and former experimental results, which in turn guide the experiments.

Experiments Description

The experiments involved two types of rover toys called 'Big Trak's', a large programmable rover and a small programmable rover Big Trak Jr. The rovers move forward, backward, to the right and the left and have additional commanding capability via a keypad. The rovers can be programmed to perform multiple instructions in a session. On each rover, one of the action keys is blacked-out. For each test, the users are to discover the behavior of the blacked-out key. The blacked-out key on all of the large rovers has the same functionality. In addition, the blacked-out key on all of the small rovers has the same functionality. Participants are divided into 10 teams of either two or three members and provided instructions to facilitate discovery of the function of the blacked-out key. The experiments are labeled A, B, C, and D. Each team is assigned two rovers and given different instructions.

Experiment A

In Experiment A the teams are given the following information about the blacked-out key:

1. The key is known as the repeat key
2. It requires a numeric parameter
3. It can only be used once in a program
4. It has no effect on instructions following its location in the program

The teams are requested to write a single hypothesis on how the repeat button might work and then to prove or disprove their hypothesis.

[1] **http://www.bigtrakxtr.co.uk/bigtrak**

[2] **http://www.bigtrakxtr.co.uk/bigtrakjr**

Experiment B

In Experiment B the teams are given similar instruction to Experiment A about the blacked-out key as follows:

1. The key is known as the repeat key
2. It requires a numeric parameter
3. It can only be used once in a program
4. It has no effect on instructions following its location in the program

Additionally, the teams are given the following three questions to consider and instructed to create multiple hypotheses before conducting any tests:

1. How do you think the repeat button might work?
2. The experiment has been previously performed with a variety of hypotheses. What do you think those hypotheses have been?
3. What do you think the designers might have considered in developing the toy?

The teams for this experiment had been asked to develop multiple hypotheses and then to prove or disprove them.

Experiment C

In Experiment C no information about the blacked-out button is provided. The teams were directed to run programs to discover the use of the blacked-out button.

Experiment D

In Experiment D the teams are informed that blacked-out button is a repeat button and given a possible hypothesis that one way in which the repeat button might work is that it repeats the entire program the requested number of times. The teams are instructed to create three possible programs prior to running them and to then run them to prove or disprove the given possible hypothesis.

Procedure

There were six teams. Each team ran two experiments. All teams were assigned to run Experiment D and run one of Experiment A, Experiment B, or Experiment C. Teams ran experiments concurrently. Three teams ran Experiment D prior to running their other experiment. All the other teams performed Experiment D as the second experiment.

Sequences

The experiments were conducted in sequential order.

Experiment A then Experiment D	Team 1
Experiment B then Experiment D	Team 2
Experiment C then Experiment D	Team 3
Experiment D then Experiment A	Team 4
Experiment D then Experiment B	Team 5
Experiment D then Experiment C	Team 6

Observations and Inferences

The following observations occurred as a result of the experiments:

1. Teams of two versus teams of three. Some teams of two found it difficult and distracting to have one person reporting and one person recording. Teams of three did not encounter this problem as the third person was able to be the liaison between the person programming and the person recording.

2. Some prior knowledge (even if partially flawed) is more useful in helping one come to the correct conclusion than no information.

3. Most teams either started with tests that were too simple, too complex or too long to be conclusive before moving to more optimally sized tests of appropriate complexity.

4. Results from a previous test tends to bias the tester towards future results of subsequent tests. For example, a team tested and discovered how Big Trak worked and then later experimented with Big Trak Jr. Some teams assumed that the results would be different based on the fact the rover was different and some teams assumed that the results would be the same based on the similarities of the two rovers, but all teams made an assumption one way or another.

5. Failure to accurately track test design and program input led to wasted time as testers questioned if they made a mistake or if they completed a valid test and observation. Some way to verify that the test steps were done correctly and that the program was entered correctly would have saved time.

6. There seemed to be two different types of testers; those that planned and then tested and those that just jumped right in and started exploring. In the end though, both approaches seemed to

be effective. In such the initial approach did not seem to affect whether the team came to the correct conclusion.

7. Testing outside of the "rules" presented in the test script often leads to quicker results. Following the "rules" presented in the test script sometimes boxed the testers in and prevented them or delayed them in learning how the button actually worked.

8. There was a degree of randomness to how quickly a team came to the correct result. The starting test sometimes stumbled onto or very close to the data needed to draw the correct conclusion. Conversely, other starting tests led some testers down false paths and wasted time. This seemed to happen more by luck than design.

9. Some of the testers were more interested or equally interested in just playing with the machine as opposed to following the script. This attitude/approach led to increased knowledge and understanding about the machine, but encompassed gathering information that was not necessarily about the specific functionality under test. This may have delayed completion of the needed tests, but would likely pay off later when other functionality needed to be tested or when/if the machine needed to be tested as a whole.

Discovery Heuristics

A number of heuristics sourced from Exploring Science [3] were examined:

1. Plausibility of Hypotheses determines extent of confirmatory bias
2. There are common aspects of an experiment that can be considered
   a) Observability: How easy/hard it was to observe the experiment. For example, using short programs made it easier to observe
   b) Distinction: How distinct were the commands in the experiment?
   c) Focusing: The selection of one dimension of the experiment
3. People have different strategies for discovering information
4. We don't have to have the full hypothesis to start
5. Designing our experiments upfront inhibits discovery but increases confirmation
6. Tracking results and experiments is beneficial
7. Prior Knowledge impacts how much time is spent hypothesising vs experimenting

Class Discovery

In addition other heuristics were discovered:

1. Developing a hypothesis prior to the start of testing seemed useful

2. Working with a small team (two or three testers) allowed teams to share ideas
3. Prior knowledge on the product seemed to lead to faster discovery
4. Listening to everyone on the team seemed to lead to faster discovery
5. Note taking of the experiments run was useful to review progress
6. Being able to simplify the test when necessary was a heuristic used by many teams
7. Running a simple test supplemented allowed rapid feedback and was useful when the test failed.

Further Work

The work pages provided to the participants had an inconsistent design, and failed to emphasize important information in all of the cases. In particular, most participants missed the instructions in Experiment D to design first and test after. Participants also experienced confusion with Experiment B as a result of the page style differing from the other experiments' instructions.

A larger sample size is necessary to accurately measure the impact of the amount of information provided to participants on their ability to discover the function of the mystery button. Additionally, with a larger sample size, it would be possible to directly compare the tasks rather than compare them via Experiment D.

When the testing groups are co-located and communicating about the tests performed, it is possible for the results to be overheard. This eavesdropping can cause a bias for the tests other groups may soon design. In the future, participants in the experiments should be located in a way such that they do not overhear or view other groups communication about the testing performed.

Some participants noted that knowing the button is called X2, and being able to see the label, further biased them. While it helped in creating a hypothesis, it hindered discovery of the function of the button when testing an alternate version of the robot.

Conclusion

The experiments conducted helped software testers to understand how we can discover information. The test teams discovered information by having a hypothesis, then designing and running tests surrounding that hypothesis. Importantly, they evaluated the information from these tests and use it to help them confirm, modify, or dismiss the existing hypothesis. The data from these tests was fed back into our hypothesis, design and test creation suggesting that exploration is an essential part of the discovery process. They observed how prior knowledge of programming and experience in testing benefited their approach to conducting experiments but also that it has the potential to bias their thinking.

They realised the value in deliberate test design. For example, tests that were easy and quick to conduct proved useful in the early stages but insufficient in distinguishing between hypotheses. Many teams realised that complex and long tests proved difficult to observe and simplified the experiments using less steps and more distinct commands.

Interestingly, the software testers observed the benefit collaboration had on the creation of multiple hypotheses. Khlar did not explore this variant in his experiments but it seemed to have a significant impact on our ability to come up with multiple hypothesis which lead to faster discovery. Future investigations on collaboration in discovery is recommended.

In software testing, the tester's role is to discover information about the product they are testing. This information is predominantly in the form of defects in the software they are testing. The hypothesis they often deliberate on is "Where is the potential risk?" It's helpful to remind them as they test the value and danger in prior knowledge; the benefit of collaboration, the importance of deliberation in test design, and if they truly wish to discover, the autonomy to allow themselves to feedback information from their tests into future hypotheses and tests.

References

[1] Bach, J. (2006). Some Useful Definitions.
Available: *http://www.satisfice.com/blog/archives/43.* Last accessed 2nd Sep 2014.

[2] Kaner, C. (2006) Developing Skills as an Exploratory Tester. (Tutorial) Quality Assurance Institute Worldwide Annual Software Testing Conference, Orlando, FL.

[3] Klahr, D. (2000). Exploring science: the cognition and development of discovery processes. Cambridge, Mass.: MIT Press.

[4] Klahr, D., & Dunbar, K. (1988). Dual Space Search during Scientific Reasoning. Cognitive Science, 12, 1-48.

Signatories

Anja	Langby	Joshua	Robson
Jonathan	Li On Wing	Lars	Sjodahl
Chris	Whitmore	Patrice	Hamilton
Dwayne	Green	Robin	Price
Felecia	Taylor	Roxane	Jackson
Griffin	Jones	Sandeep	Patil
James	Christie	Santosh	Kolure
Jason	Coutu	Sherry	Heinze
Anne-Marie	Charrett	Siavash	Solati
Jon	Hovland	Steve	Bruce
Jonathan	Clarkin	Tim	Rodoni

Back To Index

Did you do your homework before crowdsourcing your testing ?

- by Biju V Kalleppilli

Puzzled with the question? No need to panic.

Software testing is ever changing and ever evolving. Traditional testing models pave the way for many modern methods and processes. Traditional test cycles and outsourcing are not sufficient to ensure that you're delivering a high quality product adaptable enough to fit all possible customer needs. Crowdsourcing is fast spreading and is an emerging trend in the industry

There are several crowd sourced testing initiatives that had doomed incurring huge investment losses in terms of money, time and energy. Fundamentally there are three factors that influences the decision on going for a crowd sourced test practices. They are cost, Quality and time. As a development organization you should have strategy to optimize all these factors. Based on your product test strategy, thresholds of these factors will vary. If you would like to make your crowdsourcing a complete success story, please pay attention to what I am going to say.

Important steps to remember

Decide your scope

Important point to note is that you cannot handover everything to the crowd. I do not qualify this crowdsourcing model as an expert sourcing, however it should augment the in house tests that you do within your test labs where expert testers ensure the quality of the products under the supervised tests. There should be clear demarcation on the set of the activities that will fit into a crowd.

Define your target audience

Based on the targeted end users, you need to define the testing community which is capable of delivering. A new generation gaming application most likely will be used by Gen-Y crowd. So it might make sense to get into this population for your user experience feedback.

Explain what you want

You need to clearly articulate your requirement. If you want to determine design inconsistencies and usability problem areas, make it clear to your targeted testing community. You need to help your group to understand what you are looking for failure of which might lead you to unwanted and less useful outcomes. It would be a good idea to make it more creative and funny.

Describing the process and timeline

Process should detail out how this entire cycle works. This should have the chain of events starting from the project kick off till the closure. There needs to be a start and end time to the whole process. This will help to ensure the time to market need of the product. There is a danger of overloading of defect which might become unmanageable. Unless managed properly it can become chaotic and unaccountable. There needs to be a plan to handle the sudden influx of bugs. An internal development community which is active and capable of responding with a sense of urgency is an essential support system here. You cannot make the test too deep and long running as testers might be noncommittal .Need to manage the cycle properly to bring out the most productive outcome.

Ensure the communication with the crowd

It is important to note that communication involves listening as well. Less engagement and hence Lack of accountability -Unless the group is engaged, you may not get the desired output. Workers can run away any time. Encourage the sharing of information. Their reactions will give several insights into the product quality. Essence here is how many of those communications are really understood, interpreted and acted upon. Pay attention rather to what users do than what they say.

Timeline of the tests

It is advisable to keep the tests short. Longer time might test the patience of the crowd and many may retire from the scene.

My take on the future of the testing using crowd sourcing is that it would never replace the in house testing. But more and more cloud applications will be the potential beneficiaries of this. For an Ecommerce site it is possible to multiply their sales by improving the user experience. Several companies had moved a part of their mobile app testing to the crowdsourcing. This is a new area with a lot of potential yet to be untapped. It is an exciting concept that is worth exploring.

Biju currently holds the position of Quality Manager in a major enterprise software company. He has extensive experience in Software Testing & Software program management and is skilled in implementation of Lean, Agile, Design Thinking methodologies besides possessing sound knowledge in Enterprising applications.

He has successfully managed multi-cultural and multi-geography Quality Engineering teams which has proven expertise in performance, security and automation testing topics. He holds a Bachelor Engineering degree from NIT Calicut and Post Graduate Management degree from Cochin University.

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

Optimize Software Testing by Orthogonal Way

- by Dr. Sanjay Gupta

Abstract

Software Testing is perhaps the most critical part of Software Development Life Cycle (SDLC) with roughly 30-50% cost associated with it. In current scenario, the main challenge is to deliver high quality, defect free software application with very competitive time and budget window. To achieve the same, it is difficult to test an application with all the possible test conditions (at times, it is not possible and occasionally, it is not required too). Is there any way to optimize the number of test cases without compromising with the quality of deliverables and test coverage remains the key question? The optimized set of test cases should be capable to uncover the unknown defects in a software application effectively with minimal risk? Towards achieving it, one of the answer is Orthogonal Array based Test design. This article explores the importance of Orthogonal Array based test design towards test optimization and its merits.

Introduction

It is a known fact that during SDLC, number of software defects found is inversely proportional to the time. It means that in the initial state of the product development, number of defects found are more. With time, the system becomes more stable. The cost of fixing the bugs/defects increases exponentially with time. It means that if you find a defect at the later stage in SDLC, it cost more. It is essential, to find defects in the initial state of the life cycle and fix them as soon as possible.

Following software development best practices, utilizing the right tools and processes, one can deliver a high quality software product to our stake holders (It is a fact that any software products may have some residual defects at a given time). As we discussed before that prolong testing is not always possible, the key task to achieve a good coherence between optimum testing and cost/quality associated with it. In other words, you can test an application with confidence using optimum number of test cases which helps in delivering a high quality software product with optimized time and budget?

As a development & testing engineer, one will agree that once the software product is delivered to the client with proper testing and quality checks, we expect a stable application in production environment without any serious defects and praising words from the customer in the next email. Contrary to that, if a big list of the defects reported by end users in the application code. It forces team to analyze that when were these defects introduced in the code? As a software testing professional, a question always chases us “How did they slipped during testing phase?”

Let us assume that the application contains defects in production environment. The source of these defects could be any of the following

• By chance, the end user executed the portion of the software, which was never given to testing team to test.

• Test engineer is a fresher and was governed by the instructions from development team towards testing the functionality of the code. Code was not tested with boundary values/ negative test conditions which the end user has tested.

• The most common error is the mismatch in development/QA and production environment. Many times the client environment was not kept in consideration while testing the application at offshore or in QA environment.

• The defects reported by team were not monitored/ tracked properly. In other words when the defects were reported to development team and few were not fixed and a new built of the application to test still contain few old reported bugs. It is a common practice that developer says that he/she has closed the status of the bug (means fixed the problem), we go with the developers word and do not test the application for the same error. Also maintaining the XL Sheet for defect management is difficult to maintain. Using Test Management and Defect Management tools has always an advantage. If you have budget constraints, recommendations are to use open source test and defect management tools like TestLink and Mantis.

Testing, being a challenging job needs domain skills and focused dedication to test all the paths and logic of the code. The key judgment is to decide when to stop testing in a given phase? Is there any proven mechanism or technique to get a set of optimized test cases, ensuring the best test coverage of the application? Orthogonal array based test design is one answer. Based upon the data across different projects on using OATS, recommendations says that OATS will help in optimizing best test data/ Test case combinations from a large pool of possible combinations.

Orthogonal Array Testing system is extensible used in manufacturing domain. This robust testing methodology was developed by Prof. Genichi Taguchi. As software field is also recognized as engineering discipline, a successful usage is also experienced in the Software Testing area. The OATS (Orthogonal Array Testing System) technique supports the system test efforts by enabling minimum test cases to be determined efficiently which can homogeneously penetrate the application towards assuring the maximum test coverage. The details about OA and its usage can be read in details in the book, “Quality Engineering Using Robust Design” by Madhan S. Phadke.

The main philosophy of this technique is that test cases depending on interaction between test parameter will be able to penetrate an application deeply and homogeneously. Hence, you will be able to find out more serious errors in the application. Let us discuss the usage of OATS and how effectively it works.

Let us consider an application under test. As shown in Fig 1, OA based test design is applicable where a system receives multiple input parameters/values and a pre-defined business logic. The output depends on the input values and business logic.

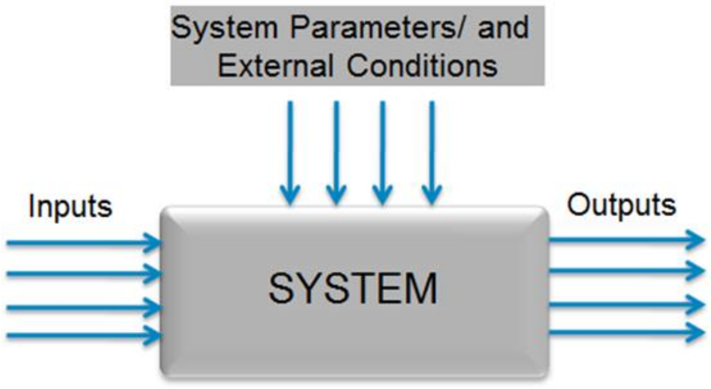


Fig.1: The applicability of OA based test design

Let’s discuss a simple scenario to Login your HR Website. You have textbox for providing user id and password. Depending on input values and business logic, a success or rejection will be decided.

Login	Password	Output
Valid	Invalid	Login Failure
Invalid	Valid	Login Failure
Valid	Valid	Login Success
No Value	Value	Login Failure
Valid	No Value	Login Failure
No Value	No Value	Login Failure

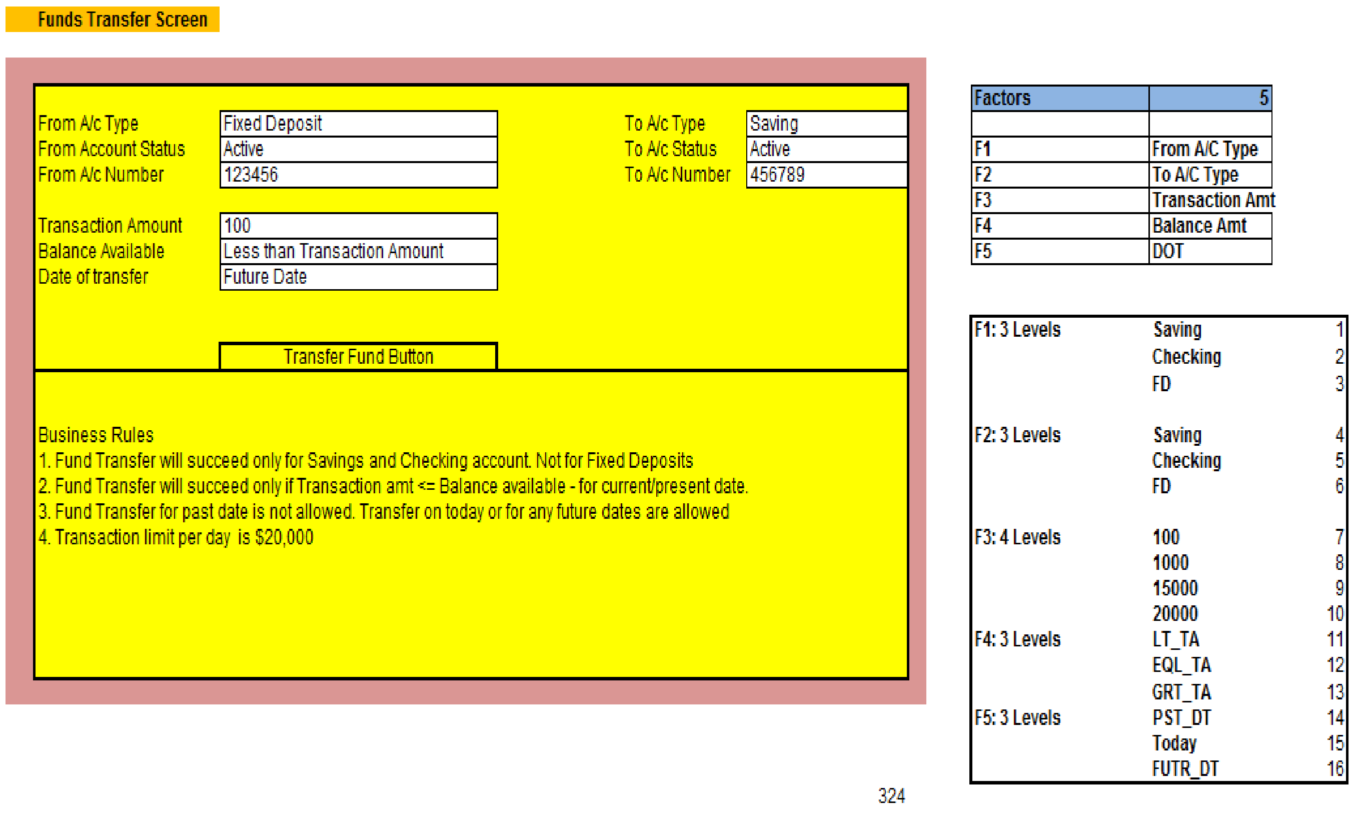
In above example, there are two inputs provided to the system.

1. Login
2. Password

In orthogonal Array terminology, the input parameters are known as FACTORS. In above example, there are two Factors (a) Login and (b) Password.

As per the discussed problem statement, Login can take three values: (1) Valid Login, (2) Invalid Value and (3) No Value. Similar way the Password field (Factor) has three values: Valid, Invalid or No Value. Number of values a Factor can have is known as LEVELS”. The above example has two Factors (Login and Password) and each factor is of three Levels. If one needs to test the above page with all permutations and combinations, Nine test cases (Multiplication of all the level values: i.e. 3*3 = 9) will be required to test the application with 100% coverage.

Let us discuss another example of a banking application "Fund Transfer" module. This module enables to transfer funds from one account to another account.



Funds Transfer Screen

From A/c Type	Fixed Deposit	To A/c Type	Saving
From Account Status	Active	To A/c Status	Active
From A/c Number	123456	To A/c Number	456789
Transaction Amount	100		
Balance Available	Less than Transaction Amount		
Date of transfer	Future Date		

Transfer Fund Button

Business Rules
1. Fund Transfer will succeed only for Savings and Checking account. Not for Fixed Deposits
2. Fund Transfer will succeed only if Transaction amt <= Balance available - for current/present date.
3. Fund Transfer for past date is not allowed. Transfer on today or for any future dates are allowed
4. Transaction limit per day is $20,000

Factors	5
F1	From A/C Type
F2	To A/C Type
F3	Transaction Amt
F4	Balance Amt
F5	DOT

F1: 3 Levels	Saving	1
	Checking	2
	FD	3
F2: 3 Levels	Saving	4
	Checking	5
	FD	6
F3: 4 Levels	100	7
	1000	8
	15000	9
	20000	10
F4: 3 Levels	LT_TA	11
	EQL_TA	12
	GRT_TA	13
F5: 3 Levels	PST_DT	14
	Today	15
	FUTR_DT	16

324

Fig 2: Fund Transfer Screen. The success/failure of a transaction depends on five input values: (1) From A/C Type, (2) To A/C type, (3) Transaction Amount, (4) Balance Amount and (5) Date of Transfer as governed by business rules.

In above example, the output of the transaction is governed by below business rules:

1. Fund Transfer will succeed only for Savings and Checking account. Not for Fixed Deposits

2. Fund Transfer will succeed only if Transaction amt <= Balance available - for current/present date.

3. Fund Transfer for past date is not allowed. Transfer on today or for any future dates are allowed

4. Transaction limit per day is $20,000

To ensure the complete test case coverage, one need to consider all the negative test case values, boundary value analysis and other input values which ensure the completeness in test coverage.

Let us assume that if the field / factor "From A/C Type" is tested with Saving, Checking and Fixed Deposit values. It ensures the complete testing for this field (No of levels= 3).

The same is with "To A/C Type" field (No of levels= 3)...

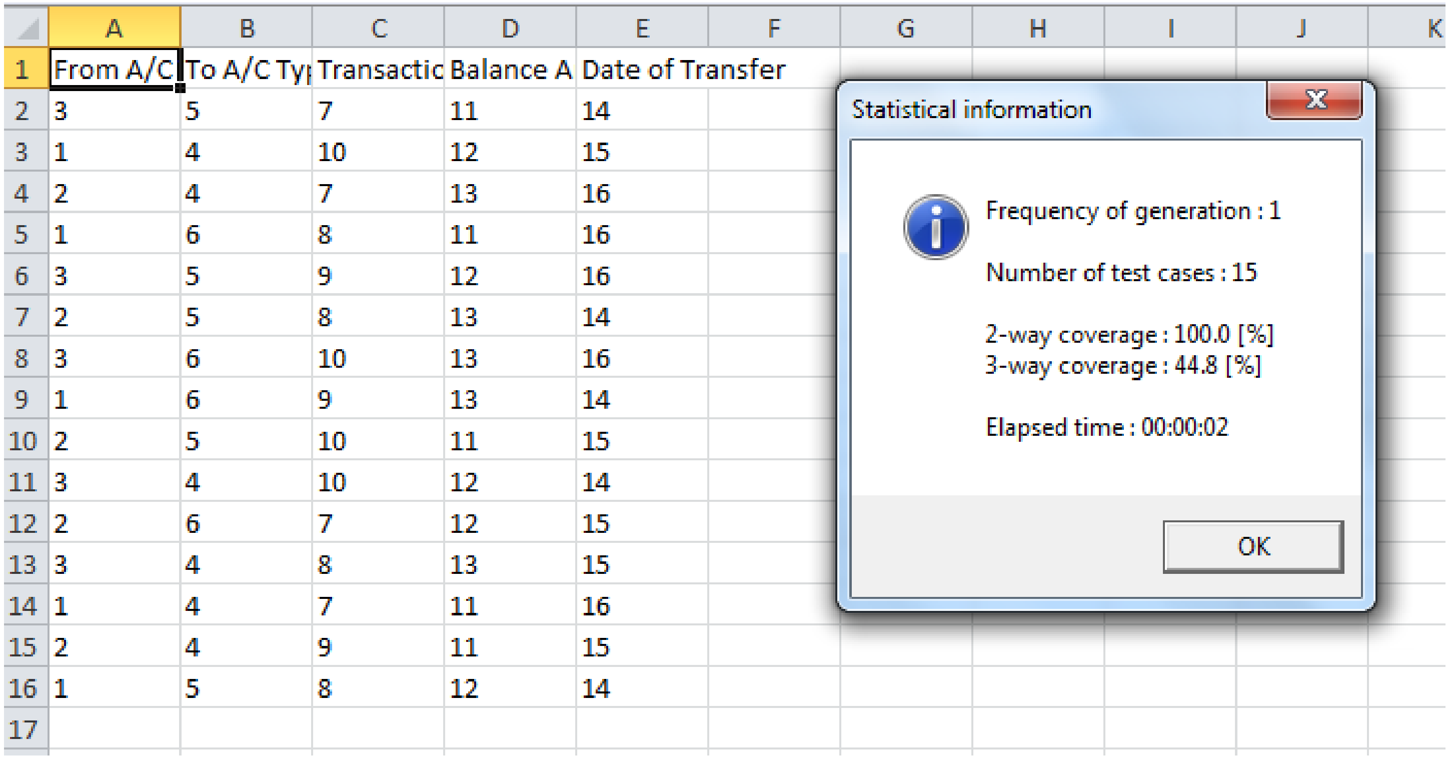
For Transaction Amount field , let us assume if we test it with $100, $1000, $15000 and $20000 values, it is complete (these values will be decided by the user, you can incorporate any other values like –ive amount, any other value like 30, 000 etc.) (No of levels= 4).

For Balance Amount field: Less Than Transaction Amount, Equal to Transaction Amount & Greater than Transaction Amount (No of levels= 3).

Date of Transfer field: if tested with Past Date, Today's Date and Future Date, we are good to go (No of levels= 3).

As depicted in Fig 2, the above example has five factors and they have respected levels. One needs to run 324 (3*3*4*3*3) permutation and combination of input values for complete test coverage. Let us analyze this problem in practical point of view. Do we really need that many test cases? Do we have time and resource to accommodate these many test cases to design and execute?

Let us see, how Orthogonal Array based test design helps in optimizing these numbers with maximum test coverage as required by you.



	A	B	C	D	E
1	From A/C	To A/C Ty	Transactic	Balance A	Date of Transfer
2	3	5	7	11	14
3	1	4	10	12	15
4	2	4	7	13	16
5	1	6	8	11	16
6	3	5	9	12	16
7	2	5	8	13	14
8	3	6	10	13	16
9	1	6	9	13	14
10	2	5	10	11	15
11	3	4	10	12	14
12	2	6	7	12	15
13	3	4	8	13	15
14	1	4	7	11	16
15	2	4	9	11	15
16	1	5	8	12	14
17					

Statistical information

Frequency of generation : 1

Number of test cases : 15

2-way coverage : 100.0 [%]
3-way coverage : 44.8 [%]

Elapsed time : 00:00:02

Fig 3: The OA based Test design output.

Fig 3 shows the output from OA based test design tool. Compare the number of test cases. Here 15 test cases ensure two way 100% coverage. This boost a confidence and helps you in deciding when to stop testing?

Once you receive the output from OA based test design tool, analyze them and modify the numbers by adding those critical test cases which you feel essential to run. You can also omit out few test cases, if you feel them irrelevant to your case.

Now if you analyses the above case closely, we just need fifteen Test cases, other were somewhat duplication. In a practical scenarios where you have a large number of test cases (in thousands) to define a regression test suite. Once a Simple OA concept coupled with the domain knowledge is used, the result shows significant optimization in number of test cases. That saves your efforts and cost without compromising the quality of testing.

Fig. 3 shows the output of the OA tool for the discussed problem. The output shows tremendous reduction in testing efforts towards testing this application. It is clear from the result that if application is tested with test cases (generated by OA along with the most essential test cases), it has covered the application with two way combinations by 100 %. Depending on the type of application, you can decide on the depth of testing (number of coverage: 3 way, 4 way etc.) and use the output. In cases, when you know few critical scenarios which are not reported by OA tool, you can add them to your test cases and make your testing more robust and reliable with a better coverage. If you see carefully all the test cases generated by the tool, it covers all those possible conditions which was practically possible. Testing the OA strength in this way has boost your confidence to utilize its power in future projects.

Can we depend completely on OA based output to release a product? Definitely Not! The recommendations will be use OA along with domain knowledge and other techniques like Dependency Structure Matrix, boundary value analysis etc. Analyze OA based test cases and ensure that it covers the complete functionality you wanted to test. Add those test cases, which are very critical to you and not covered by OA output. You can couple your OA results with suggested recommendation and come up with a reliable number of test cases which can test your application in a best way. The case studies in the real life projects have shown nearly 40-70 % effort savings when OA technique was considered as one of the major factor during Test case generation.

Summary:

In the present IT scenario by keeping the strict deadlines, budget constraints and many other factors in mind, an exhaustive testing is not viable. The effort to include all possible combinations and variations to test input parameters is generally an impracticable activity for multifaceted applications. In such a case, we mostly try to use our presumptions based on the experience to ignore certain number of test combinations. Assumptions like this may work and sometimes it may result in a poor quality product. At time, when testing efforts includes testing a large number of unattainable combinations, the Orthogonal Array based test design is very useful in coming up with best optimized set/ combinations of Test cases. It guaranties minimum one time testing the pair wise combinations of all the factors. It also drills some of the complex combinations of all the test variables.

Acknowledgement: Author is acknowledged to **Mr. Dhiraj Sinha** and colleagues for their support in developing OA based test case optimization tool and coming up this article. Thanks are also due to **Anvitha S** and **Subaya Codanda** for useful discussion/ support during development of the OA tool.

Dr. Sanjay Gupta has received Doctorate from Indian Institute of Technology, Mumbai, India. He has nearly fifteen years of IT experience with different roles as Senior Training Executive, Java/Test Consultant, Test Manager. In his current role, he is working as **Software Dev Sr. Advisor-Strategic Test Consulting and Innovation** at Dell Services, Bangalore. Sanjay is a Sun Certified Java programmer as well as Sun Certified Trainer. He has published nearly thirty research papers in international journals and presented research papers in more than fifteen international and national conferences. His current areas of research, study, and knowledge dissemination are **Java, Swings, J2EE technology, Insurance domain and testing tools like Rational Purify, Pure Coverage, Robot and Mathematica**.

His contributions towards Science and Technology has recognized him a place in **Marquis Who's Who in Science and Engineering**. He can be reached at **sanjay_gupta4@dell.com**

Planning for Effective Data Warehouse Testing

- by Wayne Yaddow

Better data warehouse testing has gained in need; qualified testers are in demand. Today there is an increase in business mergers, data center migrations, compliance regulations along with management's greater focus on data and data driven decision making. Among data warehouse testing focus areas are the ETL (extraction, transformation, load) process, business intelligence infrastructures, and applications that rely on data warehouses.

Organizational decisions greatly depend on the enterprise records in data warehouses and that data must be of the utmost quality. Complex business rules and transformation logic, built using ETL logic, demand diligent and thorough testing.

Planning for the DWH testing process

A good understanding of data modeling and source to target data mappings help equip the QA analyst with information to develop an appropriate testing strategy. Hence, it's important that during the project's requirements analysis phase, the QA team works to understand technical implementation of the DWH.

Different stages of the DWH implementation (source data profiling, DWH design, ETL development, data loading and transformations, etc.), require the testing team's participation and expertise. Unlike some traditional testing, test execution should not start at the end of the DWH implementation. In short, test execution itself has multiple phases and should be staggered throughout the life cycle of the DWH implementation (see Figure 1 below).

A key element contributing to the success of the DWH solution is the ability of the test team to plan, design and execute a set of effective tests that help identify multiple issues related to data inconsistency, data quality, data security, failures in the extract, transform and load (ETL) process, performance related issues, accuracy of business flows and fitness for use from an end user perspective.

Overall, the primary focus of testing should be on the end-to-end ETL process. This includes, validating the loading of all required rows, correct execution of all transformations and successful completion of the data cleansing operation. The team should also thoroughly test stored procedures and processes like those of Netezza that produce aggregate and summary tables.

The challenges of planning DWH tests: Listed here are just a few of many reasons to thoroughly test the DWH and use a QA process that is specific to data and ETL testing:

- Source data is often huge in volume and loaded from a variety of data repositories
- The quality of source data cannot be assumed and should be profiled and cleaned
- Inconsistent and redundancy may exist in source data
- Many source data records may be rejected; ETL / stored procedure logs will contain messages that must be acted upon
- Source field values may be missing where they should always be present
- Source data history, business rules and audits of source data may not be available
- Enterprise-wide data knowledge and business rules may not be available to verify data
- Since data ETL's are executed in multiple phases before loading into the DWH, extraction, transformation and loading components must be thoroughly tested to ensure that the variety of data behaves as expected, within each phase
- Heterogeneous sources of DWH data (e.g., mainframe, spreadsheets, Unix files) will be updated asynchronously through time then incrementally loaded.
- Transaction-level traceability is difficult to attain in a DWH
- The DWH will be a strategic enterprise resource and heavily relied upon.

Testing phases to be considered for the DWH QA Strategy

Figure 1 shows a representative DWH implementation from identification of source data (upper left) to report and portal reporting (upper right). In between, several typical phases of the end to end DWH development process are depicted such as source extract to staging, dimension data to the operational data store (ODS), fact data to the DWH and report and portal functions extracting data for display and reporting. The graphic illustrates that all data movement programs and resulting data loads should be verified throughout the end-to-end QA process.

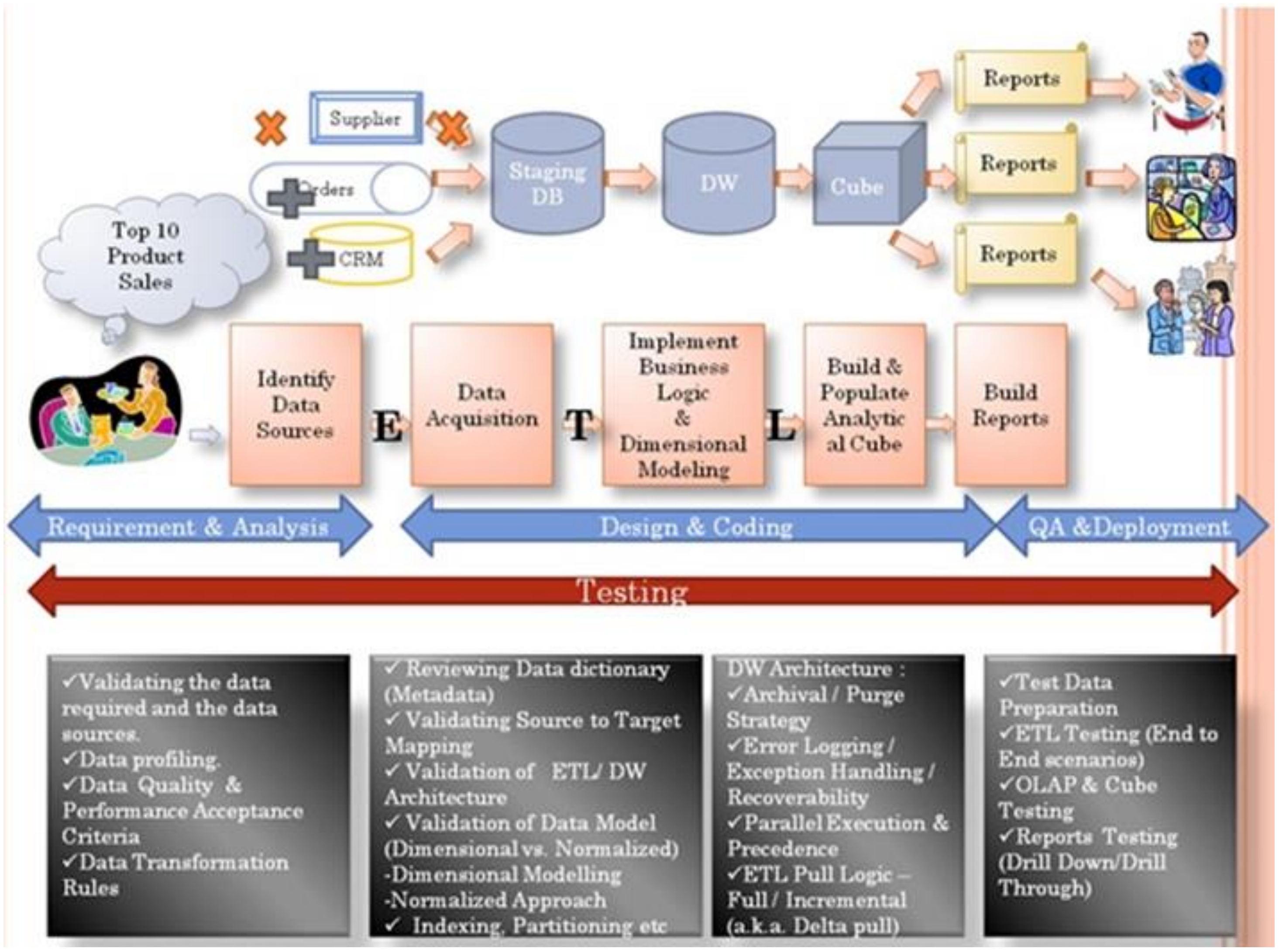

Supplier
Orders
CRM
Top 10 Product Sales
Staging DB
DW
Cube
Reports
Reports
Reports
Identify Data Sources
E
Data Acquisition
T
Implement Business Logic & Dimensional Modeling
L
Build & Populate Analytical Cube
Build Reports
Requirement & Analysis
Design & Coding
QA &Deployment
Testing
✓Validating the data required and the data sources.
✓Data profiling.
✓Data Quality & Performance Acceptance Criteria
✓Data Transformation Rules
✓ Reviewing Data dictionary (Metadata)
✓ Validating Source to Target Mapping
✓ Validation of ETL/ DW Architecture
✓ Validation of Data Model (Dimensional vs. Normalized)
-Dimensional Modelling
-Normalized Approach
✓ Indexing, Partitioning etc
DW Architecture :
✓Archival / Purge Strategy
✓Error Logging / Exception Handling / Recoverability
✓Parallel Execution & Precedence
✓ETL Pull Logic – Full / Incremental (a.k.a. Delta pull)
✓Test Data Preparation
✓ETL Testing (End to End scenarios)
✓OLAP & Cube Testing
✓Reports Testing (Drill Down/Drill Through)

Figure 1: Proposed lifecycle DWH testing process (graphic courtesy Raj Kamal, Microsoft Corp.)

Planning for QA staffing: Since a DWH primary handles data, a major portion of the test effort is spent on planning, designing and executing tests that are data oriented. Planning and designing most of the test cases requires the test team to have experience in SQL and performance testing.

Following are several tester skills that are in demand for DWH testing. QA staff for the DWH should be considered based on these abilities:

- Understanding fundamental concepts of databases and data warehousing
- High levels of skill with SQL queries – also, data profiling
- Experiences in the development of DWH test strategies, test plans and test cases – what they are and how to develop them, specifically for data warehouses and decision support systems
- Skills to create effective DWH test cases and scenarios based on business and user requirements for the DWH
- Skills and interest to participate in reviews of the data models, data mapping documents, ETL design and ETL coding; also, to provide feedback to designers and developers

Best practice DWH testing is needed as organizations seek to develop, migrate, or consolidate their data warehouses. It's vital that data and systems are tested systematically for errors, bugs, and inconsistencies before production.

One of the greatest risks to the success of any company implementing a business intelligence system is rushing the data warehouse into service before testing effectively with an experienced QA ETL testing team. Whether you are expanding your data warehouse or building one from the ground up, it is important to develop a well-planned and executed DWH testing process.

Wayne Yaddow is a computer testing professional. He worked with IBM as a Z/OS mainframe operating system developer and tester for fifteen years. After IBM, he continued his career as a QA consultant, working primarily in the financial industry in NYC with firms such as Standard and Poor's, Credit Suisse, Citigroup, and JPMorgan Chase. Wayne focused on business intelligence, data warehouse, data migration and data integration testing projects. As a contributing author to Better Software, TDWI Business Intelligence Journal and a participant with The Software Testing Professional Conference (STP), he has gained a reputation as a well-informed database and data warehouse quality assurance analyst. Most recently, Wayne teamed with Doug Vucevic to write the book, "Testing the Data Warehouse", 2012, Trafford Press.

Sharing is caring! Don't be selfish ☺

Share this issue with your friends and colleagues!

CONTINUED FROM PAGE 13...

Michael: [...] I remember once that Joshua Kerievsky asked you about why and how you tested in the old days—and I remember you telling Josh that you were compelled to test because the equipment was so unreliable. Computers don't break down as they used to, so what's the motivation for unit testing and test-first programming today?

Jerry: We didn't call those things by those names back then, but if you look at my first book (Computer Programming Fundamentals, Leeds & Weinberg, first edition 1961 —MB) and many others since, you'll see that was always the way we thought was the only logical way to do things. I learned it from Bernie Dimsdale, who learned it from von Neumann.

When I started in computing, I had nobody to teach me programming, so I read the manuals and taught myself. I thought I was pretty good, then I ran into Bernie (in 1957), who showed me how the really smart people did things. My ego was a bit shocked at first, but then I figured out that if von Neumann did things this way, I should.

Taking a closer look into practices from domain-driven design like creating a ubiquitous language, specification workshops, and so on, you will find pendants in much older literature. Eventually I have become convinced there are few really new practices out there - and many new combinations of practices that already work for decades in our young profession(s).

That said, TDD and BDD have their places in my personal development skill repertoire. There are more skills that I need to apply, like Exploratory Testing, Refactoring, incremental and iterative design, responsive design, etc. The key is to know-how first, and know-when second.

What challenges do you think project teams/organisations can face if these methods are followed blindly?

Well, a fool with a tool is still a fool. That also applies to blindly following practices. When you apply unit tests on a too fine level, you will have troubles while refactoring your code later in certain directions.

If you apply scripted scenario testing with test automation frameworks that lend their language from BDD, you will run into the problem that the maintenance of your scripted scenarios becomes a pain - especially when these rather long scenarios fail, and you want to reproduce why. (I have seen scenarios of up to 200 steps in the field.)

On the other hand, I have seen implementations of TDD, BDD, ATDD, Specification by Example, Exploratory Testing that really helped the teams to move forward. I think the key there is to create the feedback channels to improve the practices, and make them work for us, rather than blindly following something that someone somewhere at some point in time found worthwhile to try out (without understanding the contextual factors behind it).

Does implementation of these methods eliminate the need of testing or 'tester' as a role? Does that mean a programmer writing automated tests or tester writing automated test is sufficient? Does that essentially mean there is no need of manual and exploratory testing (or sapient testing to be precise)?

I believe that testing is a skill, just like programming. There are people that elegantly can apply either skill. Sometimes we call the one testers, sometimes we call others programmers, and sometimes we call them architects or developers.

Like with any skill, it helps if you know at least a bit about the skills that you need to work with. If you work in a bakery, besides the skill to put the dough in the oven, you also can benefit from knowing about the skill to knead it. The best kneader in a bakery won't be able to produce any great cookies without the skill of putting the dough in the oven. So, if you don't have the "putting dough into the oven" skill on your own, you need to work together with folks that have that skill. You need a team that can knead the dough, and that can put it into the oven.

It's pretty similar with a team that develops software. On a team you need some folks with the programming skill, and you need some folks with the testing skill. And you need to know when to apply which skill, and become aware of skills that you currently don't have on your team, and need to compensate for somehow differently.

Mostly people associate testers with the testing skill, and programmers with the programming skill. I have seen great testers that were called programmers in the company they worked for. And I have seen great programmers that were called testers. All of them were able to produce great results together with their teams because they knew how to make their knowledge about a particular skill an advantage for the team's purpose.

In the past I have been terrible at predicting anything about the future. So, it might be that at some point in time we have reached a state where all of us have learned enough about testing so that we don't need any pure testers any more. We certainly need to find out how to overcome the problem of cognitive dissonance before that. And this journey might take us a few centuries.

When it comes to exploratory testing, I don't see how we will get rid of it. In my view, with automation we automate knowledge that we already have. There will always be a demand to examine those corners of our knowledge that we don't know - and that includes also the knowledge about the things that we don't know that we don't know them. Exploratory Testing is a good starting point to put some light into these areas.

Do you believe in tester Vs programmer mindset? Do you think testers can get biased if they get involved more in development or programming?

If you dive deeper into the background regarding tester vs. programmer mindset, you will come across the concept of cognitive dissonance. In essence that means that I will be biased when testing the code that I developed on my own. The 1960s and 1970s solution to that problem was to create departments for testers, and that's probably as well when test factories started to grow for the first time.

Personally, I believe that the whole software industry would have taken a completely different turn if we kept on exploring alternative solutions to overcome cognitive dissonance. I have seen companies with no testers, where peer testing the code among developers was the way to go. This is one solution that worked for a medical devices company that created software to run in the emergency room. The first tester they hired became their first ScrumMaster.

I think we need to continue exploring those alternatives with an open mind. I also believe that we are currently putting too much interpretation into the whole “testers could be biased by being more involved with programmers” thing. If you think that your testers can become biased by being more involved with programmers, then you should also be concerned about their bias by reading the same documents, i.e. requirements documents. Let’s try to educate testers well enough so that those biases don’t lead us into trouble, and move on.

You often speak about Cognitive dissonance. What is it? How does it impact testers? Is it good or bad if we consider typical tester's mindset?

Cognitive dissonance is neither good nor bad in itself. You need to consider the context where it surfaces. Cognitive dissonance means that I will be too eagerly biased to trust the code I wrote, and not try out some of the essential test cases since that framework handles that one, or “what could go wrong here?” Testers are not necessarily impacted by it, but might if they start to work closer together with programmers. They then risk to trust the code too eagerly. Something that can be solved by proper education.

My bottom line is this: I think the traditional testing department was a solution to cognitive dissonance stemming from the 1960s and 1970s – and in fact large materials in the traditional testing community nowadays steams from 1960s to 1970s. I think there are other solutions that are more in line with today’s competitive, fast-paced world.

If you don't believe me, just compare the computer industry from the 1970s with today's. How many systems have there been? How many are there today? Why do you think the same solutions still make sense at all?

You have written couple of books on testing. Please tell us more about your books, publications.

The only full book that I have written completely on my own was ATDD by Example. I was inspired by Kent Beck's approach to discuss TDD in Test-driven Development – By Example. I took the same structure. That means that we work through two examples using the approach. Then I dive deeper into some of the connections that help learners new to ATDD to reach for the principles that are described in other books. That was the overview vision when I first started working on the book itself.

I also contributed a bit to other books. How to Reduce the Cost of Software Testing from Matt Heusser and Anand Kulkarni come to mind. I basically describe the advantages of agile testing in it. I was heavily inspired by Elisabeth Hendrickson for my chapter in that book. Literally I took her awesome presentation, and described the principles and practices for it.

There were other publications, but most of them are in German. For example for the translation of ATDD by Example I took the effort to update the frameworks, and give the screenshots a facelift. I am currently working as well on some secret writing projects that I don't dare to mention as of now. So, stay tuned. :)

What impact books have made in you? Which books would you like to recommend for testers?

I really can't answer that. I read a bunch of books since 2006. It's probably around 200 to 300 right now. There are lots of things that inspired me, and I always make a conscious effort to pick the next book to read based upon where I can learn most. That said, a book I can recommend is "How to read a book". Oh, and there is no chicken-and-egg problem involved in this recommendation. It's a great book that can help you find your approach to reading books – and that reading books from cover to cover is not the only thing to do.

What is the ideal form of influence in your opinion?

Hmmm, that's a broad question. First of all there is no ideal form in my book. Today I would formulate it this way: a form of influence that makes the influenced person follow the influencer by picking to be influenced.

That's also why great responsibility comes with great influence.

What are those skills and habits that have made you what you are today?

It's partially curiosity, and partially to get involved in so much stuff. Oh, and it's also a mixture of things that I did from age 14-26, including working in a supermarket, doing seasonal preparations for our outdoor swimming pool in the community I lived in, and being a swimming trainer and organizer of swimming events.

That said, I keep on re-inventing myself every few years. For example in 2014 I started to dive into gardening with my 800sqm of garden back at home. Over summer I became a gardening nerd. Now I hope that I know how to grow my lawn without all the stuff that used to be in there. Oh, and I also learned that I probably need to shift jobs in order to do that in a reasonable manner. That's when I realized that some outside help from a gardening professional might be worth the money. :)

What makes you an excellent tester? What would be your advice for our readers?

Keep your curiosity, and follow that. It will lead you to the right job at the right time. But maybe you will also need some patience to get there.

Thanks for your time Markus. It was great talking testing with you!

Happiness is....

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

Like our FACEBOOK page for more of such happiness

https://www.facebook.com/TtimewidTesters

Download Free NOW!

STM

NOVEMBER - 2014

YEAR I ~ ISSUE I

Tell a story, do not just report

We all make reports/presentations for/to senior management. As technical people, we are typically generous with facts. Since a picture is worth a thousand words, we spice it up with graphs. Feeling happy with the amount of details we have packed in, we are raring to make a great presentation to the senior management. Hey, wait a minute...

What are we trying to achieve? To furnish information or provide insight to assist in decision making? To report facts or tell a story? To showcase value or the quantity of work done?

I prefer story telling. Telling an engaging story that is factual, easy to assimilate, providing insights to facilitate decision making and finally showcase the value of my work. Think it is a good idea? But, what does this take?

Certainly it takes more than just dumping a lot of facts and getting into too much detail in the beginning. In fact less is more! And this is what people find difficult. Let me summarise my recent experience with my colleagues who came to me with a dense and detailed presentation that they intended to present our customer's senior management.

1. Understand target & their expectations- "Who is the audience for the story?"

Before you jump into making the presentation, understand to whom are you presenting to and what they are looking for. Typically senior folks look for outcomes and value, not just activities and facts. When you report activities done, align these against the intended objectives. And if you have issues that hinder your activities, state clearly what you need from them to resolve, don't whine! Don’t just report tons of facts, process these into meaningful information. Remember the senior management is going to ask you the question ’so-what?’ when a fact is presented, hence analyse & process these to meaningful information that can be correlated against intended objectives.

The patience levels of senior management are typically low, hence be clear about the objective, crisp in communication and hold their attention by 'outline first and detail later'. A good story needs a great plot, well-formed characters and a cogent buildup to the climax.

2. Organise the content - "Setup the chapters/scenes"

A good story flows smoothly. Start with a clear exposition of the objective of presentation, and then identify key information to be presented. Arrange the facts (content) into key sections, much like the chapters in a story creating a cogent sequence that form the story line. Then connect these information across these slides. It is the weaving of information that strengthens the story. The interrelationship of the activities, intended goals, and the final outcomes is what strengthens the story and makes it interesting.

3. Detail out the information/facts - "Choose the characters wisely'

A great plot with a cogent buildup is not enough. It is characters that breathes life in a story. In our case, the characters relate to the various facts/information. The information to be presented, their degree of detail, how objective they are, the relevancy to the context is what that makes the character contribute to the story line. Keep in mind that we are presenting to senior management who value objective information and have a high degree of impatience to wishy-washy subjective information.

4. Present the information well - "Dress up the character"

If the characters are not presented well, it is not fun reading the story. They have to fit well into the role, present themselves well, play the part for the right time and exit gracefully.

In addition to what information is presented, how it is presented makes the character play its part well. The way the information is presented include the layout, colour schemes, consistency, spelling, grammar, sentence formation, tone of voice, choice of fonts, use of callouts, using pictures in lieu of text. Short phrases, pleasant colour schemes, right choice of tone (active/passive, 1st/3rd person), appropriate fonts are what 'dresses up the character' making them very presentable.

5. Edit, re-edit - "Practice, practice..."

Much like how the characters need to practice multiple times to deliver a stellar performance, writing is iterative, requiring multiple edits, each one refining and embellishing. In every edit, refine by removing information that is superfluous. Remember 'less is more', by removing potentially noisy information. When you can remove no more, you probably have reached the end of edit cycle.

6. Read aloud - "Watch the characters perform"

Once you think you are done with the presentation, read the text aloud. In the recent case, I read some of the text in the presentation around and watched the author (my colleague) squirm! Well they could spot the issue and know what they had to modify. This works wonderfully to quickly analyse the content and spot issues. Note that you have to read text in the presentation using the right modulation for you feel good, or spot problems. This is very much like watching the performance of the characters and rapidly getting to know as to what you want to tweak.

We started with "Understand target & their expectations" focusing on what they want and now we close by playing the role of customer by reading it aloud. It is indeed amazing as to how many times I have helped refine content by becoming the customer via reading aloud.

When drab reporting gets converted into a story, the author can make the presentation come alive and engage the typically impatient senior management. Leadership is about great storytelling and when you engage in this manner, you lead the conversation. You are in control then and therefore can get your message across powerfully be it to communicate the value of your work or seek assistance to smoothen bumps you have encountered.

As engineers, we are typically left-brained rooted in logical thinking. On the other hand story-telling is a right-brained activity involving creativity and emotions.

So the next time, you make a presentation, ask yourself "Can I tell this as a story?" At the least, sit in a quiet corner and read aloud and assess if it feels good. I bet you will know what to do. If you find soliloquy challenging, do this with your best buddy.

Cheers.

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

www.talesoftesting.com

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

Advertise with us

Connect with the audience that MATTER!

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

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

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

Well, hear directly from them.

And the **Good News** is…

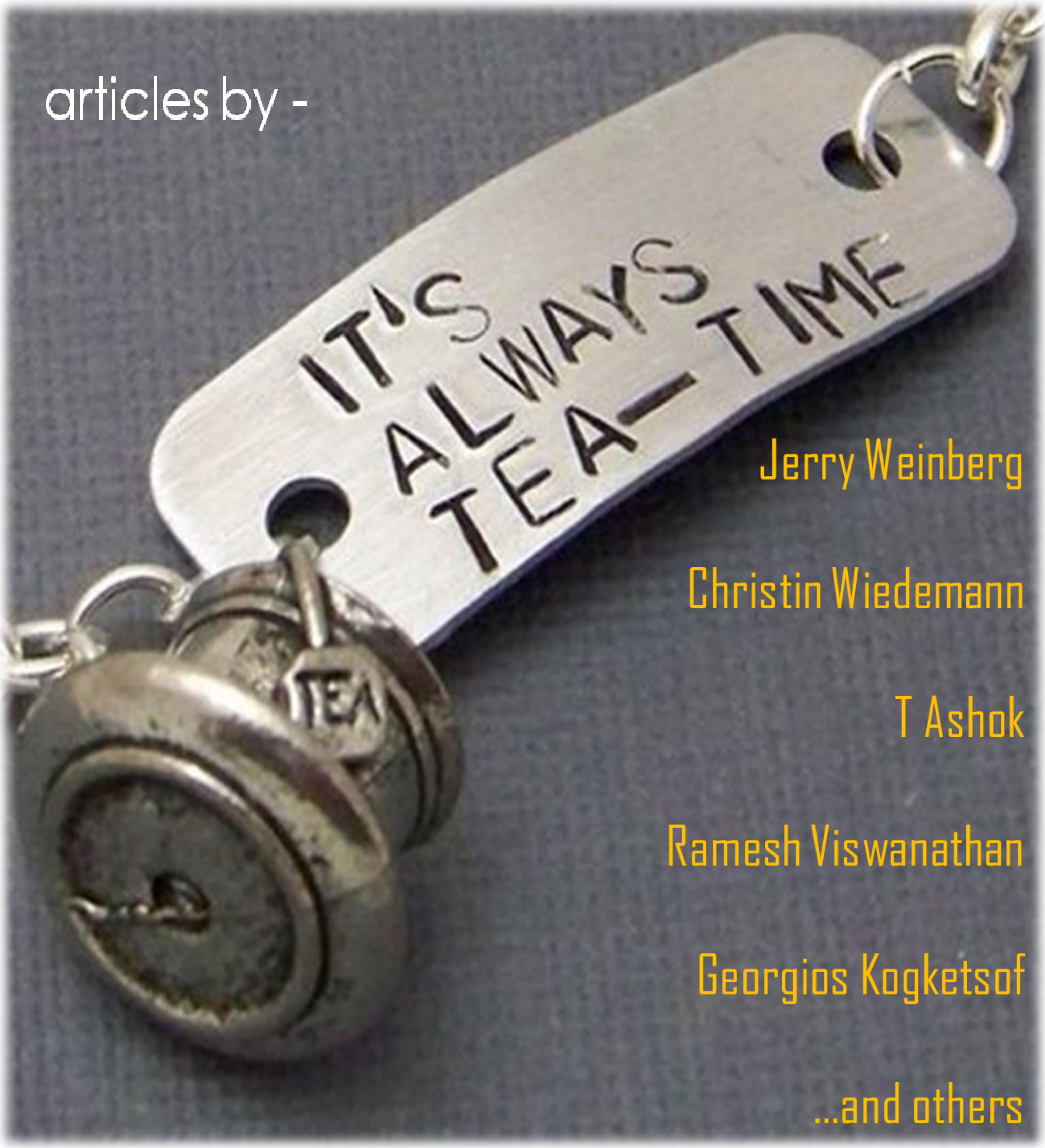
Now we have some more awesome offerings at pretty affordable prices.

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

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

in nexxt issue

articles by -



Jerry Weinberg

Christin Wiedemann

T Ashok

Ramesh Viswanathan

Georgios Kogketsof

...and others

our family

Founder & Editor:

Lalitkumar Bhamare (Pune, India)

Pratikkumar Patel (Mumbai, India)

Lalitkumar

Pratikkumar

Contribution and Guidance:

Jerry Weinberg (U.S.A.)

T Ashok (India)

Joel Montvelisky (Israel)

Jerry

T Ashok

Joel

Editorial | Magazine Design | Logo Design | Web Design:

Lalitkumar Bhamare

Cover page image – Anu @ My Dream Canvas

Core Team:

Dr.Meeta Prakash (Bangalore, India)

Unmesh Gundecha (Pune,India)

Dr. Meeta Prakash

Unmesh Gundecha

Online Collaboration:

Shweta Daiv (Pune, India)

Shweta

Tech -Team:

Chris Philip (Mumbai, India)

Romil Gupta (Pune, India)

Kiran kumar (Mumbai, India)

Kiran Kumar

Chris

Romil

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

To get a **FREE** copy,

Subscribe to our group at

Google™

Join our community on

facebook

Follow us on - @TtimewidTesters

www.teatimewithtesters.com

Give Feedback