

THE INTERNATIONAL MONTHLY FOR NEXT GENERATION TESTERS

Tea-time with Testers

JULY 2013 | YEAR 3 ISSUE VI

Jerry Weinberg
Direct Observation of Quality

Mike Talks
Crossing the Cultural Gulf

Rajesh Mathur
Context Driven Delivery : Experience Report

Nilanjan Bhattacharya
Thinking Smarter about Software Testing

Joel Montvelisky
Learning how to change your mind

T Ashok
Language shapes the way we think





How about taking that entire testing wisdom sitting right in front of JAMES BACH himself?

What if we tell you that you can take it in INDIA this year?

QAzone is glad to announce...

RAPID SOFTWARE TESTING

By JAMES BACH

This December in **New Delhi, India**

Click [HERE](#) to register

(Avail Super Early Bird discount by registering before **August 29th**)

For queries, please contact: [**events@qazone.in**](mailto:events@qazone.in)





TEA-TIME WITH TESTERS

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

Created and Published by:

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

Editorial and Advertising Enquiries:

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

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

No part of this magazine may be reproduced,
transmitted, distributed or copied without prior written
permission of original authors of respective articles.

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

Editorial



Jobs, Certifications, etc... etc.

This is true incident that happened few days back.

I was standing near my car which was getting repaired in service center. There came this big fat SUV which looked quite new but had some serious trouble in its engine, as told by its owner. After completing the formalities, guys at service center started to investigate the problem. A group of 3 junior mechanics led by one Automobile Engineer (I learned about his degree later) were trying to figure out the issue. Almost one hour passed but there weren't really able to find the reason. I was carefully listening to their conversation. Junior guys had given up and were requesting their lead to call up some guy named Harish. Lead was adamant and told them to shut up. He did not forget to remind them that he was automobile engineer and knew things better.

Another hour passed but I could see no progress. Junior guys argued and finally managed to bring Harish there. In between, I had heard that name (Harish) so many times that I was getting eager to see him and how he solves the problem. Finally I saw Harish. By his appearance, it took no time for me to figure out that he too was a mechanic. The guy came in, spent some 15 minutes and told where the problem was. I could easily notice the smile (of victory) on faces of those bunch of mechanic fellows. One of them knew that I was watching the whole thing. He came to me, smiled and said, "Fir bhi kuch fayda nahi sahab. Uske pass *certificate* hain. Bina degree walon ko kaun puchta hain?" (It's still of no use, sir. He (the engineer) has certificate. Who cares about guys who don't have degrees?)

I didn't know what to say and I too left. Needless to mention, this incident made me think around similar things happening in software testing field.

No certification = No jobs, No promotions (?)

Believe me or not, but every third message that I receive via LinkedIn asks either for guidance on (one specific) certification or dumps or some study material. The so called study group (a yahoo mailing group basically) for this certification remains flooded with request for dumps, books and 'Successfully cleared XYZ level. Hurray!' kind of emails.

It is unfortunate that very few testers seem to be concerned about their testing skills or aspire to become great at testing. Majority of them look more worried about getting such certifications done at any cost which will get them job, onsite or promotion.

But, are really *testers* responsible for this? What makes them so helpless that they don't even question the value that certification offers them? Is certification necessary just to prove that one is *aware* of testing terms and definitions? How does it imply that candidates possess great testing skills and can help organization find information it seeks about software they make?

I must mention that I am not against the idea of certification but denying someone an opportunity to prove, on basis of specific certification (or considering some *certified tester* as worth of things, without critically examining him) is dangerous and equally ridiculous...

My humble request to those who write me for guidance/help on specific certification, please read [this](#) or [this](#), before you think about it again. Guys, if becoming great at testing is all you want, I strongly recommend you to take course like [RST](#) or [BBST](#), which will help you learn testing skills that'll help you test better. Isn't that kind of value, one would naturally expect from any class/certification he takes?

In one of his [interview](#) Jerry Weinberg has said, "Testing has barely been born yet. As IT matures, so will testing. Without testing, IT will never mature." I wonder what value and maturity people see in those '40 questions' that they insist people to get certified and prefer to hire only certified testers.

In my opinion, this thing called 'certification' is becoming [Bhasmasura](#) of testing field. It's high time to kill him before he kills this field and profession. What all we need to do is, to become [Mohini](#).

Next time, if you ever get denied for not having certification, please ask recruiters, what certification *they* did to get themselves recruited. Of course, you should have great testing skills first.

See you next month.

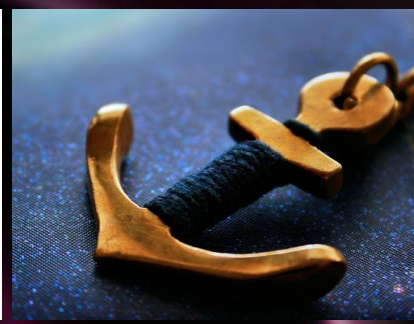
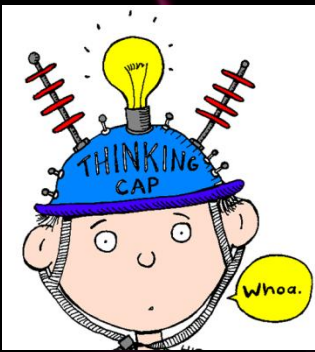
Yours Sincerely,



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



QuickLook



Testing Puzzles
by Sebi

Editorial

What's making News?

Tea & Testing with Jerry Weinberg

Speaking Tester's Mind

Crossing the Cultural Gulf- 16

In the School of Testing

Thinking Smarter about Software Testing-23

Context Driven Delivery:Experience Report- 27

A Beginner's Guide to NoSQL- 31

Raise your Anchors – Learning how to change your mind - 36

T' Talks

Language shapes the way we think - 42

Testing Puzzle – S.T.O.M. Contest

Family de Tea-time with Testers



What's making News?

- find out the latest happenings in the technology world

QASymphony releases JIRA Connector to their qTest test management solution

ATLANTA, GA—August 6, 2013

QASymphony (www.qasymphony.com), a leading developer of Quality Management solutions for software developers and QA testers, today announced the release of a comprehensive connector for JIRA Download Version 5 and higher to qTest, their [enterprise test management](#) solution. qTest provides a collaborative work environment for teams to manage requirements, design test cases, plan test execution, track defects, and generate status and quality-metrics reports. The qTest connector for JIRA combines the powerful test management features of qTest with JIRA to create a complete QA management solution.

Developed by [Atlassian](#), a San Francisco-based provider of enterprise collaboration software for product development teams, [JIRA](#) is an industry-leading project and issue management software used by more than 19,000 companies.

"JIRA is the most widely adopted tool by software developers to manage issues and tickets," says Vu Lam, CEO, QASymphony. "Recognizing the versatile nature of JIRA, QASymphony improved its qTest integration to accommodate JIRA's various applications. Our goal with this integration was to provide testers with the ability to manage the entire life cycle of a ticket from within qTest and JIRA."

The qTest connector to JIRA provides a seamless bridge that enables testers to tap into the full power of qTest's test management capabilities. Users benefit from deep levels of integration at both the requirements and defect levels and a detailed, real-time exchange of information between the two systems.

Requirements Integration:

- Import JIRA requirements tickets into qTest using smart filters
- JIRA requirements can co-exist with qTest requirements
- Changes to JIRA requirements tickets are reflected real-time in qTest
- Ability to track requirements tickets against test cases and test execution
- Test coverage reports for your JIRA requirements tickets
- Detailed traceability matrix report of requirements and test case associations

Defects Integration:

- 1-to-1 mapping between JIRA projects and qTest projects
- Direct use of JIRA credentials ensures the proper application of JIRA submission permissions
- View JIRA defects inside of qTest either as a specific ID or as part of a JIRA filter

Accessible via any web browser, the qTest online solution is competitively priced at \$20 per user per month.

Earlier this month, QASymphony announced the public beta release of qTrace 3.0, its popular screen capture tool for testers and software developers with more than 12,000 downloads. Version 3 offers full support of exploratory and session based testing. A [free download](#) of qTrace3.0 Beta is available for use thru August 2013.

About QASymphony

QASymphony is a leading provider of testing solutions that fit the needs of testing organizations at any level of maturity. Whether you are making the initial move from manual processes and need basic management help or you have processes and tools in place and are looking to enhance productivity, our test management and agile testing solutions can help you test more effectively. With offices in Atlanta, GA, Dublin, CA and Ho Chi Minh City, Vietnam, QASymphony is a software company built to revolutionize how software is tested, adopted, and supported. Empowering the QA testing teams for companies such as Silverpop, BetterCloud, Visikard and Compuware, QASymphony is a software-loving team, united by a common belief that software can be better and better tested.

Website: www.qasymphony.com

Facebook: www.facebook.com/qasymphony

Twitter: www.twitter.com/qasymphony



Software Developer's JOURNAL

new ideas & solutions for professional programmers

Check us out! We're constantly changing for you!



PREORDER



BUY NOW 

We've also prepared a special 20% discount for TTWT readers!

Enter this code while subscribing: **TTWT&SDJ**

<http://sdjournal.org/>

Tea & Testing



with

Jerry Weinberg

Direct Observation of Quality (Part 3)

A balanced measure of quality

I like to summarize my position on quality measures and quality improvement in the aphorism:

Real quality improvement always starts with knowing what your customers want.

Jim Batterson, for one, has warned me that this statement is easy to misinterpret:

"I get a little bent out of shape when you defined quality as being only from the customer's point of view, though you addressed my concerns later in the book. You and I know both know that there is a component of software quality that is invisible to the user but shows up in errors down the road and maintenance cost. There have been times when I have declined to build a system the way the user asked for it because the design did not meet my personal standards of quality/professionalism."

All too often, I see systems built by analysts with fundamental design flaws and the excuse that "that's what the user asked for." It is my contention that the user is one source, but not the only source, of business requirements, but that the analyst has the responsibility to go beyond the user in determining business requirements and especially in applying good design techniques."

I agree with Jim. Aphorisms are nice, but easy to misinterpret according to your cultural bias. The aphorism says real quality improvement always starts, not finishes, with knowing what your customers want. That means that if you don't know what your customers want, you aren't ready to assure that your "quality improvement" steps are really quality improvement steps at all.

Secondly, the aphorism states that you must know what your customers want, not what they think they want, or what you think they need. To take Jim's example, your customer probably doesn't want high maintenance costs, but quite likely doesn't understand the technical issues that will raise maintenance costs. It's the job of the professional analyst/programmer/designer to help the customer understand these issues.

But suppose the customer, after being informed, still says, "I don't care. Just give me the quick-and-dirty solution." Then your professionalism comes into play, just as it would for an engineer or doctor. If you believe that your informed customer really wants something that you cannot support professionally, then you politely excuse yourself from the project—just as a physician would do if asked to perform dangerous and unnecessary surgery. If you claim to be a professional, then you cannot work with customers who don't want professional work, as you understand it and communicate it to them.

The first measure of quality

This professional approach to quality definition is not very satisfactory to those would-be Pattern 2 managers who want customer and software engineer to toe the same line, regardless of their personal convictions. One of the first things such a software Stalin tries to do is to make measurements—usually of quantity because they have no idea how to measure quality.

Once they discover that measuring the quantity of junk doesn't accomplish anything, they start to focus on measuring errors—declaring them to be measures of quality. This is premature for a Pattern 1 organization trying to become a Pattern 2. Direct observation of customer satisfaction is needed to provide the business motivation for a true transformation.

Direct observation of customer satisfaction is not easy, just necessary. Real quality improvement always starts with knowing what your customers want, and possibly what they need or expect. That's why we'll next pursue the subject of measuring customer satisfaction.

Helpful Hints and Variations

1. When cost accounting changes, quality changes. This is so because quality is perceived value, and cost accounting changes perceived value. In a mail-order company, when the time for answering customer inquiries ceases to be "overhead" and starts being charged directly to each transaction, the perceived value of a speedy, on-line inquiry system grows. Sometimes a cost accounting change is needed to motivate people to want the right things.
2. Quality can change when a system is moved to a new environment. The first time I saw the Apple Lisa (precursor to the Macintosh) I was visiting a client in Holland. They had purchased six Lisas to see if they wanted to outfit their entire research laboratory. Unfortunately, the Lisa was hardwired for the size of American standard letterhead, not the European standard A2 paper. Some Americans thought of the Lisa as a high-quality machine. Few Europeans did. All six machines were returned, never to be seen again.

3. You can also change the quality by changing the image. This makes advertising possible—as well as con games. If quality were not in the mind of the beholder, then vaporware would not exist.
4. It's perfectly legitimate and reasonable for a company to choose which customers to shoot for, and which to ignore. If they make a wrong choice, however, it will hurt their business. They can have a "high quality product" and fail, because what they really have is "a product that would be perceived as high quality by people who don't get to perceive it." So is it "really" a high quality product? This is a bit like the question of a tree falling in the forest if there's nobody there to hear it, but it comes up often because companies do fail in this way. Saying, "But we did have a high quality product," is a way for the company's backers and employees to feel a little better about their failure.
5. Dawn Guido and Mike Dedolph say, "Perfectionism is one of the main reasons we over-engineer projects. Although perfectionists may be logically convinced that your definition of quality is correct, the emotional side of this issue is still unresolved. Knowing this doesn't help us deal with the perfectionists we work with (including ourselves)."

I agree. I will write a book on this subject—as soon as I have the right answer.

Seriously, though, a future volume will deal more with this issue. In the meantime, Mike offers a good paradigm from his martial arts instructor, Stuart Lauper. "Be happy about your progress, but always discontented enough to continue to improve your skills." This is a good measure to apply to programmers or managers who apply to you for a job.

Summary

1. To produce quality software you need to know: "What is the quality of this product, right now?" There are two different approaches to answering this question—the direct approach and the indirect approach.
2. For many people, "quality" is such an ambiguous term that nobody could be against it. Ultimately, everyone has the same definition of quality, and it goes something like this: "Quality is whatever I like."
3. Quality is relative. Quality is value to some person. Value is what are people willing to pay to have their requirements met. Perfectionists whose strongest desire is to find the one right way will not be satisfied with this relative definition of quality. But, then, perfectionists won't be satisfied with anything, so forget them.
4. Many software organizations today are so overloaded with quality problems that they are no longer coping effectively with their business of developing software. Yet many managers fail to recognize the relationship between this overload and quality problems.
5. Managers also fail to recognize the relationship between their own actions and the results they're getting.
6. Every software problem is a quality problem. The Zeroth Law of Quality says, If you don't care about quality then you can meet any other requirement. If you don't have to control quality, you can control anything else.

7. Quality is the most direct software measure we can find, and the only direct way to measure quality is with the people whose ideas count. The ultimate political question is, therefore, Who gets to control the definition of quality?
8. Power corrupts. And what power corrupts most thoroughly is the ability to make meaning of observations. When software developers control the definition of quality, they may do well in the short run, but they will ultimately drive their customers to the first competitor they can find.
9. The measurement of the absence of errors is not the same as direct measurement of quality. Real quality improvement always starts with knowing what your customers want. That's the only direct measure of quality.

Practice

1. Advocates of function points claim that one of their advantages is that they measure the size of an application from the user's point of view of functions provided. Present some arguments that would support this point of view. Present some that might invalidate it.
2. Quality can also depend on when you count the customers. Over what period of time does their valuation count?
3. We can contaminate the direct measurement of quality, by the process we use to select the people used to determine quality. For example, early users of a product are not typical of later users. Nor are early non-users typical of later non-users. Nor can the developer's employees be considered typical. How would you go about choosing "typical".

[Back To Index](#)



Tutorial on 'Software Test Attacks for Mobile and Embedded Devices'

Today's expectations for many software testers include addressing mobile and embedded devices. Unfortunately for many companies, churning out complex or critical mobile and embedded applications while keeping pace with emerging technologies is fast becoming the norm rather than the exception it was just a few years ago. Competitive pressures place a burden on software testing resources to succeed with shortened project schedules, minimal strategic planning and/or staff new to mobile and embedded software.

In this tutorial, **Jon Hagar** and **Jean Ann Harrison** will provide specific in depth test attacks aimed at uncovering common mobile-embedded software bugs.

Find more about this tutorial [HERE](#).



Jon Hagar



Jean Ann Harrison

Biography

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



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

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

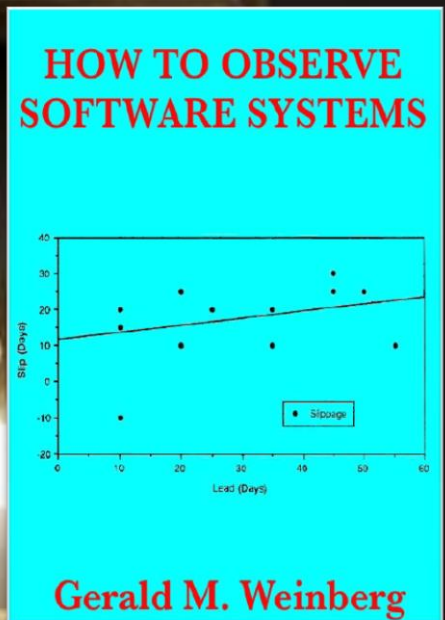
To know more about Gerald and his work, please visit his Official Website [here](#).

Gerald can be reached at hardpretzel@earthlink.net or on twitter [@JerryWeinberg](#)

HOW TO OBSERVE SOFTWARE SYSTEMS is one of the most famous books written by Jerry.

This book will probably make you think twice about some decisions you currently make by reflex. That alone makes it worth reading. "Great to understand the real meaning of non linearity of human based processes and great to highlight how some easy macro indicator can give info about your s/w development process." An incredibly useful book. Its sample can be read online [here](#).

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



TTWT Rating: ★★★★★

A photograph of a green, teardrop-shaped pendulum bob hanging from a thin wire. The bob is positioned over a surface of light-colored sand. In the sand, directly beneath the bob, is a circular pattern of concentric, slightly raised ridges, resembling a ripple in water or a sand mandala. The entire scene is framed by a dark blue border.

Speaking Tester's Mind

- straight from the author's desk

Crossing the cultural gulf



- Mike Talks

When I started off on my first software job, I worked in a company of a hundred employees who were mostly white, male. Oh we had a few women on staff to answer the phones, but when it came to managing, testing, and coding, it was men all the way.

Fast forward 15 years, and I have recently finished at Kiwi bank, where, on reflection, I was in the minority on the team being *"the white male"*, not the majority.

As software and the areas it touches expands and continues to grow, so the people inside the industry continue to diversify. Change which expands our family can only be a good thing, but it doesn't come easily.

Ironically, most of the "old guard" in the software industry were probably more the school outcasts, seen as nerdy or misfits. I spent a lot of time as a 10 year old in the early 80s going to libraries trying to find about computers, and how they worked. My teacher tried to discourage me (*yes, discourage me from using the library*) saying computers would not change the world. *I'll let you be the judge of who won that argument.*

When the first home computers came out, a small gang of us would swap software and coding tips, try out each others' machines, share magazines and books which had listings to *"write your own software"*. We'd show versions of these games that we'd customised. Our friendship was based on sharing this passion together with other *"common pursuits"* such as Star Wars, James Bond, Doctor Who. We had black and Asian friends from school who were included, but let's be clear, it was very much a boy's club. *Girlz were not allowed!*

And so, we've grown up, got into the software industry, but part of the Boyz Club mentality remains. And ironically because of this, the school outcasts have unwittingly created a clique of their own.

Take a look below, you'll see one of my desks at home – it's probably a desk that's typical amongst developers ...



Right next to it is a stack of comics and the complete set of Monty Python scripts. *Yes, this should be very familiar.* Indeed, maybe the following video of programmers as perceived in the 80s is too close for comfort ...



This is one of the things which attracted many of us to the IT industry, it was almost a way to continue those *Abbot Beyne Boyz Computer Clubs*, to take on a job, but like Peter Pan keep hold of that childhood passion for life. There is a great quote though about such people, *"I love his boyish charm but I hate his childishness"*.

When I started blogging about testing I wanted to do away with some of the business lingo, and make the ideas accessible, and so as much as possible I used my cultural frame of reference.

One of the articles I was most proud of was one I wrote after a tense week called **"*The Kobayashi Maru Of Office Relationships*"**. This was a piece about how like the infamous test in Star Trek, our relationships in the office don't have easy win situations, but then looks at the dynamics of this.

I later shared with a couple of friends at work, who although they enjoyed it, said a lot of it went over their heads, as they *"didn't really do Star Trek"*. Sure enough, the article is loaded with references to TV and films which I adore – my friends who enjoy the same things really got a lot out of the piece, those who didn't found it didn't have the same power.

And this is the problem of the *Boyz Club of IT* it's no doubt unintentional, but there is a very real cultural clique within IT. Like *"The Kobayashi Maru Of Office Relationships"* we use cultural folklore and metaphors which are comfortable for us, because their use is so incredibly powerful.

We need metaphors and examples, because they take concepts and make them more powerful and real in our minds. We might talk for instance of love, but that's just a nebulous concept. But if we talk about an example that we all know about, suddenly the talk of love becomes something so much more real. We talk of *"Romeo and Juliet"* and the metaphor stirs powerful feelings within us ... unless that is we don't know the story. Then it becomes something baffling and confusing to us.

I have to be careful here, because I'm about to break my rules to give an example – a *very nerdy example*. But in Star Trek: The Next Generation (*now if you're not a fan, please bear with me*), there is a superb episode called *"Darmok"* which deals with this very theme. Humanity encounters an alien race which it just cannot understand – all communication with them feels garbled.

So the alien captain kidnaps the human captain Picard, and they go to a planet where they are all alone except for a terrible monster. The two captains have to team up to survive. The alien captain hopes that by forcing the two of them to unite against a common foe, they will learn to forge a friendship and see their commonalities.

He's right. Proximity does lead to understanding. It turns out this alien race talks excessively using metaphors, metaphors that the human race doesn't understand. Put together like this Picard learns the stories behind their language and thus learns not only about their culture, but how to share his Earth culture with them.

Uncle Bob Martin wrote a **superb piece** in the aftermath of the Dongle-gate discussions (triggered by a developer who as mentioned *"his boyish charm spilled into childishness"*) on *"is there sexism in IT"*, in it he talks about the common stories he often uses to tell our *"IT tales"*. Whilst they can feel a lot of

fun for some of the core audience, in actual fact people outside these tales feel alienated. And when he found that out, he was horrified.

And this is where a lot of problems stem from. A metaphor or instance which is powerful to one group, sometimes can be alienating to another. My Kobayashi tale worked well, using a lot of shorthand to fellow film geeks, but in the end the core tale about *"working well with others, championing and challenging"* deserved a wider audience, not just film students (*who by the way can be male AND female*). This is why when I revisited it for my book *The Software Minefield*, I tried to broaden its aspects as much as possible, and read the piece with both my "Star Trek fan" filters on *and off*.

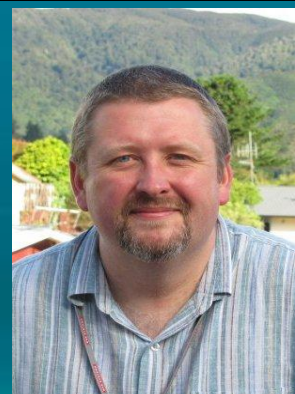
My friend Gizelle (*a fellow sci-fi fan, you don't have to be male to love sci-fi*) did a video assignment on *"Stereotype Threat"* which I've also found enlightening. In it she plays a female engineer, who feeling intimidated by her office, feels like she has to almost *"androgenise"* herself and dress and act more male to fit in as *"one of the boys"*. When we stick to our Boyz Club of IT culture too much, that's what we do, we are saying that women can participate only if they *"become one of the boys"*, in essence what Gizelles character feels compelled to do.

So should we feel ashamed of the IT culture that's built around us? No, as my friend Jo Foster points out, *"stories are also handy shortcuts - tell the story once, and then you just need to refer to it, rather than having to tell the whole thing again and again"*, they do have a power, but only in the context of our audience. We have to be aware when members of our audience in our team or workplace do not understand those stories, because they will feel alienated by them. It should be our job to try and build bridges, not to build walls.

As the Star Trek episode "Darmok" highlights, to understand, we need to have an environment where we can mutually share cultures. That means us listening to others tales as much as talking of our own, and always respecting passion in whatever form.

Sharing is important. I learned to come from the *Boyz Club of IT* where only guys were in the allowed to be member to my views today, not because it was "politically correct" and forced upon me, but because I worked with people of different gender, race, culture. And I realised that a lot of differences were merely cosmetic, because they all needed respect, space and a chance to prove themselves as much as I did. Don't be afraid to share, invite them to come laser tag with you, but if they invite you to Zumba or karaoke, give it a go as well.

There is no single IT culture. IT is multicultural. We can either choose to build walls or build bridges. Choose to build bridges.



Mike Talks is a test manager in Wellington, and self-confessed movie nut. Always on the lookout to be educated as much as to educate, he is also the author of the LeanPub book 'The Software Minefield'. <http://leanpub.com/TheSoftwareMinefield>

Back To Index 

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

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

Well, the wait is now over . Introducing...

“The Guiding Star”

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



E-mail your question on –

theguidingstar@teatimewithtesters.com

Please Note :

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



12th International Conference on Software QA and Testing on Embedded Systems

**MORE
TESTING**

**MORE
QUALITY**

**MORE
EMBEDDED**

This year we offer a special program: **3 Keynotes, 2 Tutorials, 8 Tracks**, from renowned companies. Visit our website -www.qatest.org- to **discover the Programme!**

QA&TEST, the international Conference on Software QA and Testing on Embedded Systems, will be held on 29, 30 and 31 October in Bilbao, Spain, and gives you an excellent opportunity to increase your business network and establish contact with others professionals of the industry.

The main objective of QA&TEST is present the last technological developments in Software Testing and Quality Assurance, and showcase successful best practice to reduce costs and give companies a lead in global competition.

www.qatest.org

SOFTWARE

Organiser



Sponsor



October 29 · 30 · 31

2013

Bilbao · Spain

A photograph of a classroom scene. In the foreground, the backs of several students' heads and shoulders are visible as they sit at their desks. They are all raising their right hands, with index fingers pointing up, in a gesture of wanting to answer a question or participate. The students are wearing colorful shirts: light blue, red, orange, and green. In the background, a large, dark chalkboard is filled with faint, illegible chalk writing. The entire image is framed by a thick black border.

In the school of Testing

for your better learning & sharing experience



Thinking smarter about software testing

a guided tour with some of the world's most renowned thinkers

- Nilanjan Bhattacharya

[Edge.org](#) is a website where some of the most renowned intellectuals discuss ideas and express opinions. Every year, the Edge, asks a question which is answered by a large number of leaders in various scientific fields. The [question](#) in 2011 was "What scientific¹ concept would improve everybody's cognitive toolkit?" There were 165 responses from eminent thinkers. In this article I have listed the concepts which are most relevant to software testing.

How to read the links

All the responses to the Edge question are available on the web. I would recommend you first read the links I have provided before reading my commentary. [It is important to read each link – the description of each concept will change the way you think.](#) I recommend you take printouts of the web pages and highlight relevant portions. Some of these articles are heavy reading. If you wish you can read one or more every day. When reading the links, try to put aside any preconceptions you might have about software development.

[If you are a software tester, this will make you smarter!!](#)

[Wicked Problems](#) (Read the hyperlink before reading ahead)

Creating software can be challenging. Software development is definitely a wicked problem. Developers and testers are good at solving [difficult](#) problems. [Wicked](#) problems require a different mindset. I like to think that a smart developer has already tried to solve a 'wicked problem'. This makes testing even more 'wicked'.

The Name Game (Read the hyperlink before reading ahead)

The best example of the 'Name Game' is the word 'test' (related to software). There is no one in the world (including grandma) who would admit to not knowing the meaning of the word 'test'. In reality the word 'test' is highly misunderstood (in the context of software). It is important to understand what 'test' really means. Another word which is a victim of the 'Name Game' is 'agile'.

(See the related concept: [The Dece\(i\)bo Effect](#))

Experimentation (Read the hyperlink before reading ahead)

If what we do isn't 'testing', what is it? It's experimentation. This isn't being pompous. As in real life, when you are trying to determine what works (or may not work), you need to experiment. The agile movement has created practices which make software increments available in a few days. There is no reason not to start experimenting immediately and all the time. We would be better off if we realized that 'user stories' (as used in agile) are experiments. (Note that most non-scientific people use the word 'experiment' in a negative way, as if it is something frivolous as opposed to being deliberate. See the related concept 'Science' and the 'Uselessness of certainty' to remove that misconception.)

Gedankenexperiment

In the case of software, unlike physics, you *can* conduct physical experiments (on the software). However, the gedankenexperiment is a great concept to stimulate thinking and can be an alternative or precursor to physical experiments. It's another alternative to the heinous word "test".

The Double-Blind Control Experiment

It's worth understanding double-blind control experiments from [Wikipedia](#).

While the previous concepts of experiments and Gedankenexperiment are powerful alternatives to 'test', a 'control experiment' is a much more specific, rigorous procedure. One of the concepts that refer to the software development process is Timo Hannay's concept, "The Controlled Experiment" (this is the only concept not available online). Mr. Hannay explains that online companies like Google, "don't anguish over how to design their websites.

"Instead they conduct controlled experiments by showing different groups of users until they have iterated to an optimal solution. (And with the amount of traffic those sites receive, individual tests² can be completed in seconds.) They are helped, of course, by the fact that the Web is particularly conducive to rapid data acquisition and product iteration. But they are helped even more by the fact that their leaders often have backgrounds in engineering or science and therefore adopt a scientific-which is to say, experimental-mind-set."

Although this is a great example of a controlled experiment, the more powerful concept is that the entire process of software development can be viewed as a series of experiments (see Roger Schank's concept 'Experimentation'). Why do we only selectively perform controlled experiments? Can't we conduct such experiments all the time? When we don't have access to customers, can we use proxies such as in-house staff, or even developers and testers?

Cumulative Error (Before you forget, read the hyperlink before reading ahead)

While Jaron Lanier is referring to the flow of information in high end financial systems, software development itself is an elaborate game of 'postman'. The agile development community has acknowledged this problem by declaring the importance of verbal communication over "messages" inscribed on paper. Unfortunately, some traditional teams³ haven't caught on and continue to play "telephone", without realizing it.

I have often seen software developers and testers struggle with this cognitive fallacy in vain. Mr. Lanier explains this beautifully,

"The illusion of Platonic information is confounding because it can easily defeat our natural skeptical impulses."

The virtues of negative results

Related to the concept of "experimentation" is the concept that "Failure liberates success". In the world of software development, the idea of testing (evaluating/passing judgment), and the idea of logging defects (someone creating something 'defective'/'faulty') has rubbed people the wrong way. In the testing community many have written about how to log defects so as not to offend. The software development community could benefit if engineers realized that failure is a cornerstone of progress/science.

The agile movement has at least partially recognized this scientific concept – short of equating failure with progress. The idea more common in agile is to limit the scope of failures. While agile retrospectives result in learning from failure, I don't think failure is actively used as a tool for learning, i.e., engineers don't intentionally make software fail to understand flaws in design⁴. Good software testers have been doing this for a long time.

You can show something is definitely dangerous, but not definitely safe

This concept should put an end to the angst surrounding defects in the software development world (defects found during development/before release to the market). Since you cannot **prove** that the software being developed will cause no harm (to users), you continue to find ways in which it can cause harm (defects).

Tester redeemed!

Uncertainty

Uncertainty is a **virtue** when dealing with complex systems, as long as it is quantified. With complex systems there is limited value in pursuing precision as a goal in itself. It is more valuable to qualify uncertainty. This concept is different from uncertainty in software development/agile, where uncertainty is about what to expect next or the learning from what didn't work. It is more relevant to software testing when you try to quantify unknowns.

The uselessness of Certainty

We don't **need** 'scientifically proven.' We need enough information which will allow us to make decisions and act. The agile movement has successfully thrown off the shackles of Tayloresque management and adopted a personal, direct approach based on valuing people and their skills and personal needs. However, in traditional testing and management, the word "scientific" often shows up as the preferred approach. "Scientific" implies well documented and detailed, and engineers who follow instructions. While in most teams there is no compulsion on rigorously following instructions, i.e., engineers are allowed to question what was documented, the bigger point is lost, viz., **there is no value in certainty!!** Will the software crash when it has to connect to eight servers? Even if you "test" it with eight servers, you can't be certain. If it doesn't crash, you could be proven wrong, when the environment changes. The key challenge in software development is to correctly qualify your level of uncertainty.

Science's Methods Aren't Just For Science

Science is a way of thinking, of making better approximations of how things are.

For software developers, software development is a hard technical skill with the discipline of math or "science". Agile has greatly broadened that outlook to include communication, especially with customers, teamwork and an overall understanding of how people work. Despite those gains, "testing" is probably still seen as "fuzzy" by software developers, while most **traditional** testers would balk at equating testing with "science".

Software engineers (testers) would greatly benefit by using scientific methods to make sure customers perceive value in their products (or they should quantify the uncertainty about risks to that value).

Scientific methods **don't need** to include conducting experiments. They could be activities like a "journal club" or applying different ways of thinking such as vertical, horizontal, or learning to take meaningful notes when observing software behavior.

(See the related concept **Science**)

Q. E. D. Moments

You need to experience the divine moment of proving a logical or mathematical problem. This may not be straight forward if you don't have a math/science background. When developing and testing software, i.e., determining value and what might go wrong, you can conduct investigations to validate your hypothesis. In the long term, it's still worth pursuing the goal of experiencing a QED moment.

I have listed only a small number of the concepts discussed in the book and website. Most of the concepts are worth reading and will make you think smarter. Many of the concepts can be applied in everyday life, not only to work.

Notes:

¹From the website: "The term 'scientific' is to be understood in a broad sense as the most reliable way of gaining knowledge about anything, whether it be the human spirit, the role of great people in history, or the structure of DNA. A "scientific concept" may come from philosophy, logic, economics, jurisprudence, or other analytic enterprises, as long as it is a rigorous conceptual tool that may be summed up succinctly (or "in a phrase") but has broad application to understanding the world. "

²Although the use of the word 'test' in this context is fine, do cross-reference the concept "Nominal fallacy".

³This is not a blanket judgment against teams not using 'agile'. Teams not using agile have effectively promoted similar structures to improve communication.

⁴While TDD is a great concept, starting with failed tests is different from the concept explained here.

[Back To Index](#)



Nilanjan Bhattacharya manages a test team in the R&D lab of IBM Security Systems in Singapore. He has 18 years experience working with software development and testing teams across U.S., India and Singapore, working on both consumer and enterprise products.

Nilanjan's detailed profile is available at sg.linkedin.com/in/nilanjanswmanager/. His twitter handle is @nilanjanb and he blogs at: <https://swtestmanager.wordpress.com/>

Context Driven Delivery

(Experience Report)

- Rajesh Mathur



Scenario:

Couple of years ago I was brought in to a project whose objective was to change an enterprise wide internal system. The proposed system was replacing a legacy system and was expected to be a COTS (Commercial off the Shelf) Product. There was a due diligence process in place to ensure that the proposed solution is a fit for purpose and that it has sufficient quality of the COTS product.

There are few important facts about this project that need to be considered before we learn more from the experience:

1. Cost of project: Few millions USD
2. Due Diligence activity status: Completed with 'limited business inputs'
3. Product status: Needed 'few' customization.
4. Development team at client side: New
5. Testing Team at client side: New
6. Status of business requirement: Ambiguous, Incomplete and some still in 'Work in progress' mode
7. Testing Strategy: The product was being customized in three separate phases. Each phase had its own SDLC and the whole process was iterative. Testing was aligned with each phase and planned in a way that when the first phase of product customization completes, test preparation for next phase

begins. Same for the next phase that it was to begin with whole SDLC including requirements, development, test, and defect fixes etc. Once all phases complete, a big-bang integration was to happen followed by an end-to-end Integration test by testing team followed by an end-to-end UAT.

8. Risks: If delayed for a certain period, this medium size project could be in critical path of a much larger multi-million dollar program resulting in loss of revenue and massive loss of reputation.

The Problem Statements:

1. Everything started on a positive note. However, right in the middle of phase 1 development, the development supplier discovered that there were hardware infrastructure dependencies on another supplier and that these dependencies will impact the schedule of first phase. Eventually, the first phase slipped by almost 2 months. Not only that, it had a roll-on impact on the whole schedule.

2. Impact on testing:

Test execution for phase 1 delayed. The delay had a knock on effect on phase 2 test preparation.

Phase 2 test execution got delayed due to additional scope and much longer preparation time. This directly impacted testing estimates, test team member count and schedule.

Phase 2 resulted into a disaster due to discovery of more than 150 defects. This was obvious as the vendor was still trying to deliver on time making their development team prone to more mistakes and their testing team was not able to cope with the time pressure. Unfortunately it also impacted client testing team as they were discovering more and more defects in acceptance testing and previously fixed defects were failing in subsequent releases.

Phase 3 scope increased due to multiple defect fix cycles and additional defects found in the fixed releases.

The Solution:

If we had not thought of a practical approach to resolve all issues before deployment of Phase 3 release, I wouldn't have been writing this experience report today.

Context Rules:

I was keen on resolving the issue. Hence, the first thing I did was to make a vendor visit offshore. The objective was not to screw them over open issues, but to see

1) Why there are so many issues and

2) If we can help resolving some of them.

Second thing we decided was to establish a war-room. We knew that traditional testing process will not work in this context. So we changed the approach.

We made each project team member to sit at a single place (which we called war-room) so that they can communicate more, email less, collaborate more, and integrate more. We got vendor's representative, our business analyst, business users, system analyst, testing team and project manager all at one place. We got a huge white board installed. We listed all features on the white board, the test coverage by individual testers, related defects and fixing-plan of those defects.

We started telephonic discussions with the vendor's team in India twice a day. The vendor representative on site was already there to help the team coordinate and resolve any communication issues. We started daily 15 minutes stand-ups so that people can raise their concerns and get them resolved right there and then.

A plan was prepared assigning a specific area to each tester and the developer. They performed a large number of tests pair-testing.

The big white board also had a 3 week plan running on it. So, everyone was aware what the exact status was. Vendor was told to define a delivery plan so as to increase the output and instead of monthly releases, they were asked to deliver weekly smaller releases so that they can be tested and fixed quicker. We constantly prioritized defects and high risk features.

At the end of the day, not everything worked out well; but we were in a much better shape than we anticipated in the beginning. We had relatively low number of high severity defects when the product was deployed for UAT. The number of defects met exit criteria and we got an assurance by the vendor that there was an action plan to fix even those deferred defects.

In all, testing team found few hundred defects in what the vendor called an off-the-shelf product, which needed 'just slight customization'. Had we not changed our approach...!



Rajesh Mathur is Test Delivery Manager at Hong Kong's premium airline Cathay Pacific Airways. He manages testing of multiple projects under Airline Operations and Cargo domain. With 17 years of experience in Software Testing, Rajesh has lived and established his career in countries like the US, the UK, India and Hong Kong, and has worked as a tester, test lead, manager, programme manager and test delivery manager for many high-profile multi-million dollar projects.

Rajesh is a Fellow of Hong Kong Computer Society and a member of Australian Computer Society. He has been active on the software testing arena for a long time and has been actively working on strengthening the testing community & the practice of software testing through trainings, mentoring and by speaking at conferences. He is a well known speaker and writer on testing topics. He regularly writes at <http://www.dogmatictesting.blogspot.com> and can be followed at twitter at @rajeshmathur.





Taking
a break?

a click here
will take you there



From Special Desk

contributed by S.D. Journal this month

A Beginner's Guide to NoSQL

This article aims to explore the basic ideas and principles about noSQL databases. noSQL caters to database admins, programmers, coders, web devs, etc.

Let's say you've decided to set up a website or an application. You'll obviously need something to manage the data. Yes, that's right, a database. So, what is it going to be? MySQL, MS-SQL, Oracle or PostgreSQL? After all, nothing can be as amazing as a good old RDBMS that employs SQL to manage the data.

Well, allow me to introduce to you an entirely unique and unconventional Database model – NoSQL. Just like every other fine article out there, we too shall begin with...eh....disclaimers!

NoSQL stands for not-only-SQL. The idea here is not to oppose SQL, but instead provide an alternative in terms of storage of data. Yet, for the obvious reason that most users are well versed with SQL, many NoSQL databases strive to provide an SQLlike query interface.

Why NoSQL?

That's a valid question, indeed. Well, here are the reasons:

- **Managing Large Chunks of Data:** NoSQL databases can easily handle numerous read/write cycles, several users and amounts of data ranging in petabytes.
- **Schema? Nah, not needed:** Most NoSQL databases are devoid of schema and therefore very flexible. They provide great choices when it comes to constructing a schema and foster easy mapping of objects into them. Terms such as normalization and complex joins are, well, not needed!
- **Programmer-friendly:** NoSQL databases provide simple APIs in every major programming language and therefore there is no need for complex ORM frameworks. And just in case APIs are not available for a particular programming language, data can still be accessed over HTTP via a simple RESTful API, using XML and/or JSON.
- **Availability:** Most distributed NoSQL databases provide easy replication of data and failure of one node does not affect the availability of data in a major way.
- **Scalability:** NoSQL databases do not require a dedicated high performance server. Actually, they can easily be run on a cluster of commodity hardware and scaling out is just as simple as adding a new node.
- **Low Latency:** Unless you are running a cluster of a trillion data servers (or something like that, give or take a few million of them), NoSQL can help you achieve extremely low latency. Of course, latency in itself depends on the amount of data that can be successfully loaded into memory.

SQL ideology

Basically, NoSQL drops the traditional SQL ideology in favor of CAP Theorem or Brewer's Theorem, formulated by Eric Brewer in 2000. the theorem talks about three basic principles of Consistency, Availability and Partition Tolerance (abbreviated as CAP), adding that a distributed database can at the most satisfy only two of these. NoSQL databases implement the theorem by employing Eventual Consistency, which is a more relaxed form of consistency that performs the task over a sufficient period of time. This in turn improves availability and scalability to a great extent. This paradigm is often termed as BASE – implying Basically Available, Soft state, Eventual Consistency.

NoSQL Data Models

Some of the major and most prominent differentiations among NoSQL databases are as follows:

1. Document Stores
2. Hierarchical
3. Network

4. Column-oriented
5. Object-oriented
6. Key-value Stores
7. Triple Stores

Document stores

Gone are the days when data organization used to be as minimal as simple rows and columns. Today, data is more often than not represented in the form of XML or JSON (we're talking about the Web, basically). The reason for favoring XML or JSON is because both of them are extremely portable, compact and standardized. Bluntly put, it makes little sense to map XML or JSON documents into a relational model. Instead, a wiser decision would be to utilize the document stores already available. Why? Again, simply because NoSQL databases are schema-less, and there exists no predefined form for an XML or JSON document and as a result, each document is independent of the other. The database can be employed in CRM, web-related data, real-time data, etc. Some of the most well known implementation models are MongoDB, CouchDB and RavenDB. In fact, MongoDB has been used by websites such as bit.ly and Sourceforge.

Hierarchical Databases

These databases store data in the form of hierarchical relevance, that is, tree or parent-child relationship. In terms of relational models, this can be termed as 1:N relationship. Basically, geospatial databases can be used in a hierarchical form to store location information which is essentially hierarchical, though algorithms may vary. Geotagging and geolocation are in vogue of late. It is in such uses that a geospatial database becomes very relevant, and can be used in Geographical Information System. Major examples of the same include PostGIS, Oracle Spatial, etc. Also, some of the most well known implementations of hierarchical databases are the Windows Registry by Microsoft and the IMS Database by IBM.

Graph Network Databases

Graph databases are the most popular form of network database that are used to store data that can be represented in the form of a Graph. Basically, data stored by graph databases can grow exponentially and thus, graph databases are ideal for storing data that changes frequently. Cutting the theoretical part, graph database has perhaps the most awesome example in the likes of FlockDB, developed by Twitter to implement a graph of who follows whom. FlockDB uses the Gizzard Framework to query a database up to 10,000 times per second. A general technique to query a graph is to begin from an arbitrary or specified start node and follow it by traversing the graph in a depth-first or breadth-first fashion, as per the relationships that obey the given criterion. Most graph databases allow the developer to use simple APIs for accomplishing the task. For instance, you can make queries such as: "Does Jonny Nitro read Data Center Magazine?" Some of the most popular graph databases include, apart from FlockDB, HyperGraphDB and Neo4j.

Column-oriented Databases

Column-oriented databases came into existence after Google's research paper on its BigTable distributed storage system, which is used internally along with the Google file system. Some of the popular implementations are Hadoop Hbase, Apache Cassandra, HyperTable, etc.

Such databases are implemented more like three-dimensional arrays, the first dimension being the row identifier, the second being a combination of column family plus column identifier and the third being the timestamp. Column-oriented databases are employed by Facebook, Reddit, Digg, etc.

Object-oriented Databases

Whether or not object-oriented databases are purely NoSQL databases is debatable, yet they are more often than not considered to be so because such databases too, depart from traditional RDBMS based data models. Such databases allow the storage of data in the form of objects, thereby making it highly transparent. Some of the most popular ones include db4o, NEO, Versant, etc. Object-oriented databases are generally used in research purposes or web-scale production.

Key-value stores

Key-value stores are (arguably) based on Amazon's Dynamo Research Paper and Distributed hash Tables. Such data models are extremely simplified and generally contain only one set of global key value pairs with each value having a unique key associated to it. The database, therefore, is highly scalable and does not store data relationally. Some popular implementations include Project Voldemort (open-sourced by LinkedIn), Redis, Tokyo Cabinet, etc.

Triple stores

Triple stores save data in the form of subject-predicate-object with the predicate being the linking factor between subject and object. As such, Triple Stores too are variants of network databases. For instance, let's say "Jonny Nitro reads Data Center Magazine." In this case, Jonny Nitro is the subject, while Data Center Magazine is the object, and the term 'reads' acts as the predicate linking the subject with the object. Quite obviously, mapping such semantic queries into SQL will prove difficult, and therefore NoSQL offers a viable alternative. Some of the major implementations of Triple Stores are Sesame, Jena, Virtuoso, AllegroGraph, etc.

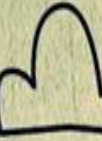
Summary

So, what now? Well, you've just been introduced to NoSQL. However, does this mean that you should make the switch to it from SQL? Perhaps. Or perhaps not. The answer varies from situation to situation. If you find SQL queries way too much to cope with, chances are you'll find NoSQL equally difficult. However, if you're looking for a more flexible alternative and do not mind getting your hands dirty, you should definitely give NoSQL a spin! The choice, obviously, is yours! Happy data managing to you!

Sufyan bin Uzayr is a 20-year old Freelance writer, graphic artist and photographer based in India. Sufyan has been extensively involved in the field of graphic design and web development, and he has also developed apps for the mobile platform. Currently writing for two print magazines and six blogs, Sufyan is also the Editor-in-Chief of Brave New World, a contemporary electronic journal. Visit Sufyan's website at www.sufyan.co.nr or his e-journal at www.bravenewworld.in You may also mail him at sufyan@live.in



are you one of those
#smart testers who
know d taste of #real
testing magazine...?



then you must be telling your friends about ..



Tea-time with Testers

Don't you ? 😊



Tea-time with Testers !

first choice of every #smart tester !



testing intelligence

- *its all about becoming an intelligent tester*



an exclusive series by **Joel Montvelisky**

Raise your Anchors – Learning how to change your mind

I was listening to a report on the radio the other day on my way to work.

They talked about a new study showing that the most common reason behind doctors' mistakes is their inability to look for alternative explanations once they made their initial diagnosis on a patient, even when new information came up pointing at other possible causes for the patient's condition.

The article explained that this behaviour is related to the concept of **anchoring**, developed by the Nobel Price laureate **Daniel Kahneman** together with Amos Traversky back in the 70's (and written extensively by **Dan Arieli** in his "Irrational" book series).

In plain words anchoring means that once you believe in something you will find it hard to move away from this idea. Sometimes even after being presented with hard data showing that you may be wrong.

I read (and enjoyed) the “Irrational” book series by Arieli, but only when I heard about anchoring in the context of doctors’ malpractices did I realize that testers are not immune to this behaviour too.

Anchoring in the context of QA & Testing

We tend to make up our minds on someone based solely on our (erroneous?) first impressions.

You may decide a tester is smart only because she said something intelligent during the first 5 minutes of the work session you had with her.

Or vice-versa, you may decide a tester is dumb because he was too quiet or maybe because he made a “trivial” mistake during your first work session together (and you failed to notice he was timid and even somewhat nervous by the whole situation).

The same goes for applications or features we need to test.

You may think that a feature is “clean of bugs” because you hardly found any important issues in your first round of sanity testing.

Then you may let important bugs slip out because you “lowered your testing defences” and failed to check thoroughly during the following deep-down testing round.

What about the bugs we find?

(This happened to me a number of times in the past)

You report a bug and for some reason you decide to categorize it as a Critical defect.

Then, even if people come to you with good and valid reasons to lower its priority you find it hard to agree with them. You go on to provide your own reasons for keeping this issue as a Critical bug (even if in your heart you know their reasons are valid).

BTW, you can say that same about reporting a bug, and later on not agreeing that the behaviour was not a defect in the first place...

Giving a “scientific” name to your irrational behaviour helps

After all these years I am happy that my sometimes-irrational behaviour got a scientific name, it definitely beats the feeling that I had of sometimes being an egocentric fool. Now I understand that this is caused by an intrinsic flaw in my basic human design 😊

More over, I believe that once we are aware of the cause it is easier to fight and even get over these behavioural issues.

Techniques to fight your anchoring

There are a number of things you can do if you suspect that you are suffering from Anchoring in your QA tasks.

In order to keep it short I will only explain a handful of them:



Defocus – Defocusing basically means to create a perspective on what you are testing by looking at the whole picture and understanding the complete “why” of your application, instead of only concentrating on the details or the “what” of the specific feature you are about to test.

I’ve read many articles that explain defocusing, but I personally like this one by Anne-Marry Charrett (Maverick Tester) that explains why she is the **Queen of Defocus**.

Walking the dog – This is one of my favourite techniques.

As a tester, once you feel that you have run all your testing scenarios (both formal and informal), and that you are ready to provide your testing feedback back to your team, WAIT.

Take some time to switch your context by “walking the dog around the block”, by going to drink coffee in the kitchen, by running some other testing task, or simply by doing anything that will clear your mind.

After this, get back and review all the testing tasks you ran, look for things that you may have missed or paths that you forgot to follow before.

Only once you are sure there are no new areas or paths to test you should go ahead and “mark” your tasks as done.

You’ll be surprised how many “small but important things” you might have forgotten to do in the first place.

Swap testing – This is a basic but useful technique.

Don’t trust only yourself to perform any task. Whenever possible, schedule time for someone else from your team to run some tests on the app you are testing.

It’s not that you want another person to run the same scenarios you just did.

What you are looking for is for the inputs and the new scenarios other testers may check that you did not even think to review in the first place.

Another common technique closely related to this one is peer-testing.

5 critical bugs – This is mostly a planning technique.

Even if you “think” that the feature you are about to test will be clean of bugs (this may be because it is an easy feature, or because you trust the specific developer, or because of any other esoteric reason), take the time to define the 5 most critical bugs that may be hiding in this feature. Then use these theoretical bugs to plan your tests around them.

BTW, it is even better if you can brainstorm about these bugs with other people in your team.

Fighting the up-hill battle against anchoring in testing 😊

Once we understand that we may be “a little biased” or that we have a tendency to look at the world in “a certain way” it is easier to correct this behaviour.

Have you had experiences where you’ve been “anchored” in your testing?

How did this affect your work and what did you do to correct it?

Share these testing-war stories with us by adding them as comments, and help other testers fight against their natural anchoring behaviour.




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

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

Today Joel is the Solution and Methodology Architect at [PractiTest](#), a new Lightweight Enterprise Test Management Platform.

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

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



Call for Articles !

Have you got something to say?

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

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

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

Submit your articles, stories, thoughts around software testing.

now its your chance to be heard...!

Click [HERE](#) to read our Article Submission FAQs !

T ' Talks



T. Ashok exclusively on software testing

Language shapes the way we think

Language is not just for writing, it plays a significant part in our thinking process. It has been discussed widely as how "Language shapes the way we think" (Read and listen to one of these at <http://blog.ted.com/2013/02/19/5-examples-of-how-the-languages-we-speak-can-affect-the-way-we-think/> for an interesting TED talk blog & talk).

Once we are comfortable with a language, we "think" in that language. For example, we form sentences in the mind to understand, form narratives to describe, think in terms of if-then to discern logic, create command sequences to create actions and so on.

Language is made of the syntax 'the rules' and the content 'the semantics'. The syntax i.e. rules shape the way and the depth of understanding of the content.

Language allows us to (1)Describe a story "Understand" (2)Breakdown the problem "Simplify" (3)Setup clear boundaries "Baseline" (4)State the purpose "Goal" (5)Organize our thoughts "Plan" (6)Issue instructions to get things done "Action" (7)State what has happened "Report" (8)Document stuff so as not to forget "Remember".

Now let relate these to 'testing'. Look at the 1-7. Sounds familiar? This is what we do when we commence testing - Understand, Simplify, Baseline, Setup goal, Plan, Action/do(Test), Report and Remember for future.

Now let us see how the language which we typically use to document/write shapes how we think.

(1) Describe a story "Understand"

Understanding is a key element to good testing. To understand whose need(s) we are trying to satisfy and the value expected, it is critical to think like an end-user. We often use the term "think from the user point of view", but it is easier said than practiced. To enable a deeper and better understanding of the system a persona-based approach i.e. of Describing the behaviour and the associated attribute(s) in first person as if the end user is describing it from his/her point of view enables you to put yourself in the shoes of the end user, empathise and therefore understand better.

(2) Breakdown the problem "Simplify"

Any non-trivial thing is presumed complex. A true hallmark of good understanding is de-mystification, that of making it simple. From a language perspective, it is about summarizing, of describing in short sentences and not exceeding a paragraph.

(3) Setup clear boundaries "Baseline"

A clear baseline as to what-to-test (I.e. requirements/features) is necessary to ensure clarity in what we need to and that we have indeed covered i.e. evaluated completely. Using an imperative style sentence that is short and precise forces us establish a clear baseline. For example a customer requirement may be stated as "That the system admin shall be able to ..." while an example of technical feature is "That the system shall provide" Note that a descriptive or a narrative style is a strict no-no here.

(4) State the purpose "Goal"

Testing is about ensuring the needs of the various end users delivered via the various technical features do meet their expectations. Not only it is necessary to clearly outline the needs as a clear baseline, it is equally necessary to ensure that the expectations are well stated.

This implies that the baseline has to be qualified with a criterion that is indeed objective. For example "That the system admin shall be able to do 'blah' within 'x' seconds on these 'y' devices". I.e. a short imperative sentence with a qualifier that is objective.

(5) Organize our thoughts "Plan"

This is one of the things that we do most frequently in daily life, the To-do list. The way we do this is to list down activities in a numbered bulleted list in sequential order (based on time). The language that we typically use is in first person using an imperative style. The method that we use to think is in terms of bullets with a imperative style heading with a narrative style to describe the plan of each To-do action in detail.

(6) Issue instructions to get things done "Action"

This is where we come up with scenarios to test. What I have observed is most often a narrative style description that describes the actions to be performed, the data to be used, and the method of validations to assess correctness.

From a language perspective it is necessary to be action oriented here I.e. describe each scenario as a command and the associated expected result in single sentence and then describe the steps to perform. For example "Ensure that the system does/does-not 'foo' when 'bar' is done". First be clear of what you want to accomplish before you jump to how-to-do.

(7) State what has happened "Report"

Now this is the fun part as reporting can describe multiple facts that are all connected, leading to complexity and confusion. From a language perspective, reporting is describing outcomes arranged by elements across time with associated detail and therefore the sentence to describe these can turn out to be inherently complex. This is applicable to report a defect, to report the outcome of a test cycle, to report final test results, to describe learning etc.

To ensure clarity of thought, it is necessary to partition the description first in terms of summary and description, then partition the detail into smaller elements and then describe each element along multiple dimensions.

In the case of defect report we describe a short synopsis of the problem and then describe with multiple elements like 'detailed observation', 'method to reproduce', 'environments observed in' etc. In the case of test report, we commence with a summary and then describe the various each element in a section with different dimensions to describe in detail the elemental information as possible subsections.

(8) Document stuff so as not to forget "Remember".

This is really the free form part, the part that we jot down everything we observe, learn from past. This is one part that we cannot stick to one style of syntax. This is a mixture of all styles mentioned above and beautiful mixture of terseness with detail.

The structure of sentence matters to the way we think, understand, perceive. Ultimately the content, semantics matters, but truly the structure matters and syntax is a great guide. A guide who can shape how you think, help you stay on a path of clarity. Syntax used in a rote matter may be seen as restrictive, but clearly it marks the path of clarity. Use it. Use it wisely.

It matters how we write/document. Clarity is truly a function how you describe. Remember language shapes the way you think or how it makes others think!

Enjoy!



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](#)

Testing PUZZLES

by Sebi



Claim your **Smart Tester of The Month** Award. Send us an answer for the Puzzle and bellow b4 5th September 2013 & grab your Title.

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

Puzzle

In bug bounties some of the high rewards were given for web applications that use SMS confirmations.

Detect and list SMS functionalities for Google, PayPal, Facebook and Meraki. Give workarounds where possible if there are restrictions per country or other similar ones.



[Back To Index](#)



Biography



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

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

He tweets as @testalways.

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



Every Tester

who reads Tea-time with Testers,

**Recommends it to friends and
colleagues .**

What About You ?

in ne>xt issue

articles by -

Jerry Weinberg

T Ashok

Joel Montvelisky

Bernice Ruhland

Silvio Cacace

Shashidhar Penumala

our family

Founder & Editor:

Lalitkumar Bhamare (Mumbai, India)

Pratikkumar Patel (Mumbai, India)



Lalitkumar



Pratikkumar

Contribution and Guidance:

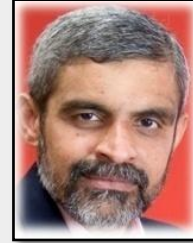
Jerry Weinberg (U.S.A.)

T Ashok (India)

Joel Montvelisky (Israel)



Jerry



T Ashok



Joel

Editorial | Magazine Design | Logo Design | Web Design:

Lalitkumar Bhamare

Core Team:

Dr.Meeta Prakash (Bangalore, India)



Dr. Meeta Prakash

Testing Puzzle & Online Collaboration:

Eusebiu Blindu (Brno , Czech Republic)

Shweta Daiv (Mumbai, India)



Eusebiu



Shweta

Tech -Team:

Chris Philip (Mumbai, India)

Romil Gupta (Pune, India)

Kiran kumar (Mumbai, India)



Kiran Kumar



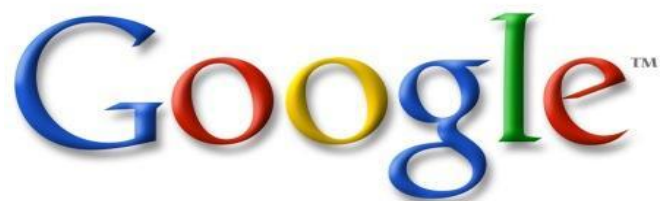
Chris



Romil

*// Karmanye vadhikaraste ma phaleshu kadachna /
Karmaphalehtur bhurma te sangostvakarmani //*

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



Join our community on



Follow us on



www.teatimewithtesters.com

