THE INTERNATIONAL MONTHLY FOR NEXT GENERATION TESTERS

Tea-time with Testers

SEPTEMBER 2015 | YEAR 5 ISSUE VII

Articles by –

Jerry Weinberg

Wayne Yaddow

Jan Jaap & Miranda Kerpel

Rahul Verma

T. Ashok

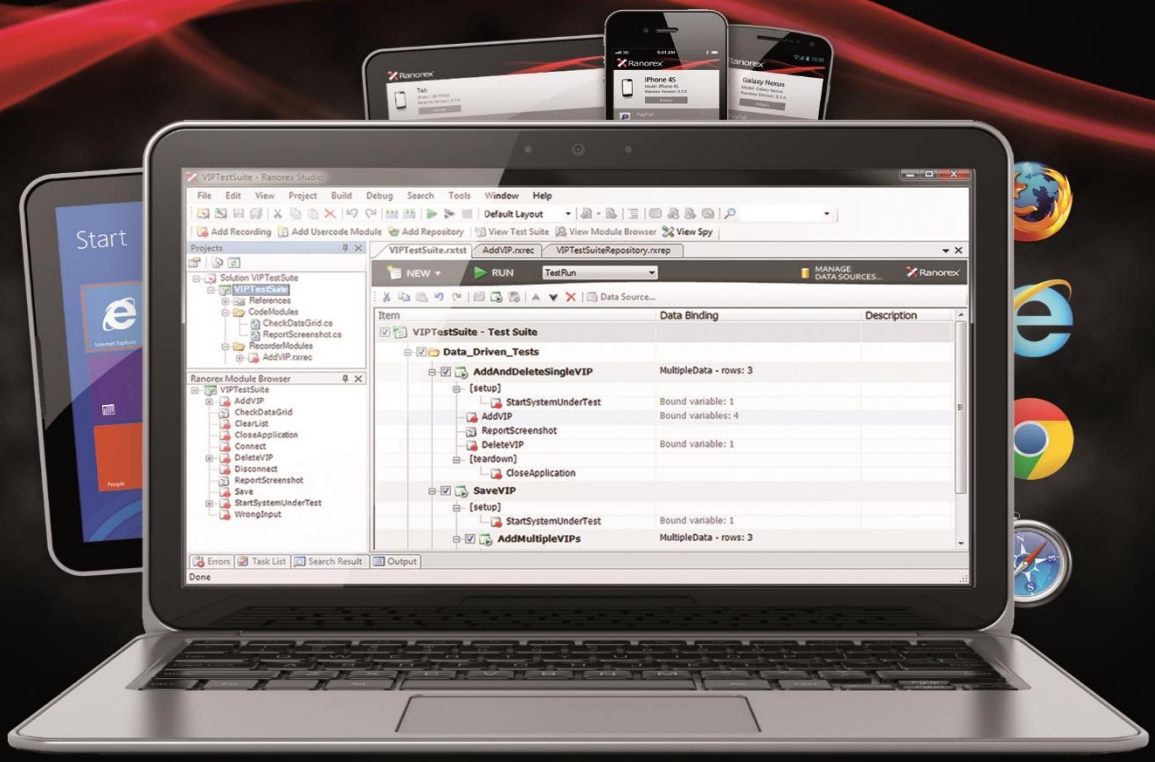
Tobias Walter

Thanh Huynh

Over a Cup of Tea with Dr.Cem Kaner

Ranorex®

Automated Testing of Desktop. Web. Mobile.



- Robust Automation
- Broad Acceptance
- Seamless Integration
- Quick ROI

Why Use Ranorex

www.ranorex.com/why

Award-winning test automation tools provide seamless testing of a wide range of desktop, web and mobile applications.
Android | iOS | HTML5 | IE | FF | Chrome | Safari | WPF | Flash/Flex | Silverlight | Qt | SAP | .NET | MFC | Delphi | 3rd Party Controls | Java

Sign Up For A Free Demo Today

www.teamqualitypro.com

GET A FULL VIEW INTO ALL YOUR DATA

- Inhouse/Offshore contracts
- Defects
- Test phases
- Software Development
- Reports
- Payroll/ Contracts
- All projects
- Notifications

- Project Progress
- Team Analysis
- Top 10 Best / Worst
- Cost Measurement
- Defects / Testing

TAKE THE FIRST STEP IN SAYING:

YES TO:

- Real time reporting
- Instantaneous data gathering
- Objective, vetted data
- Comprehensive data
- Data that mirrors your process
- Data that is always current

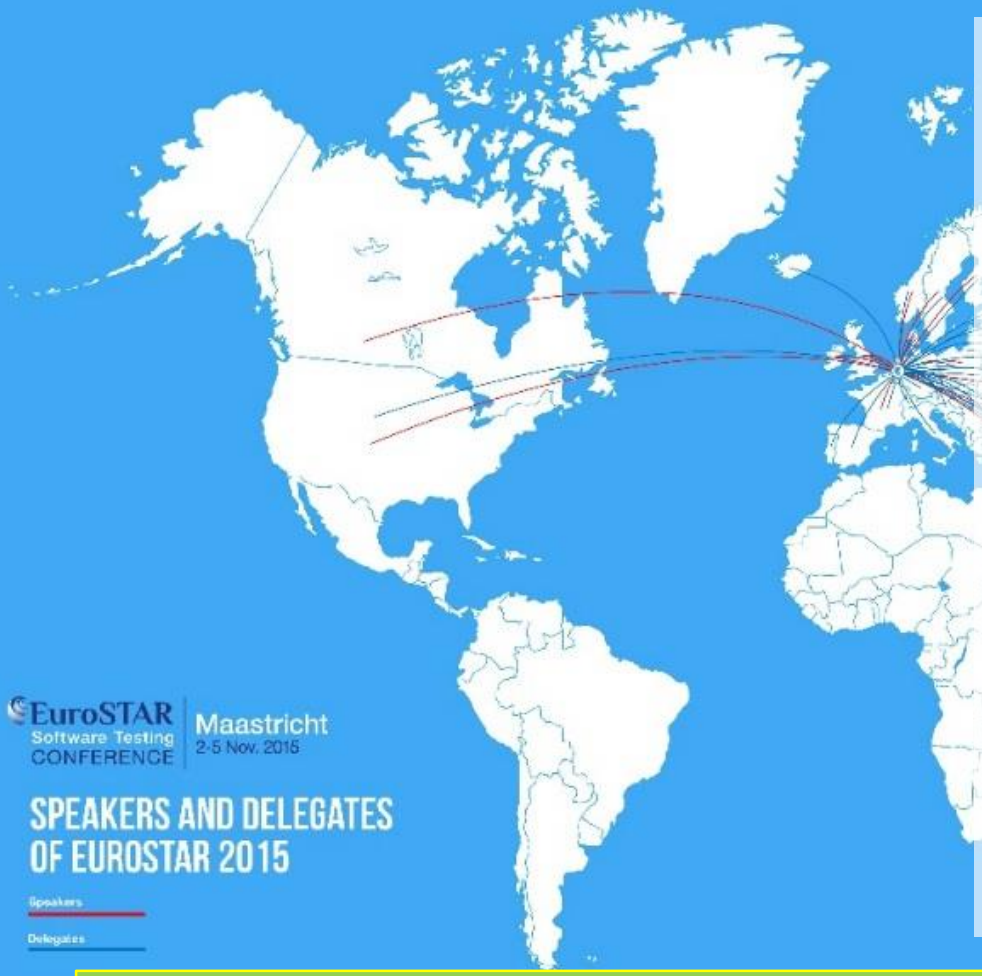
NO TO:

- Manual reporting
- Data warehouse projects
- Subjective data
- Missing data
- Not aligned data
- Wrong time frame data

FUEL YOUR PASSION
EXPLORE NEW BOUNDARIES
NEVER STOP TESTING

EuroSTAR Software Testing CONFERENCE | Europe's #1 Software Testing Conference **Maastricht 2-5 Nov**

In just under 4 weeks, testing professionals will be gathering from all over the world to the MECC, Maastricht for the 23rd annual EuroSTAR Software Testing Conference.



Are you a professional software tester? Are you looking to be inspired in 2015? EuroSTAR is the place for you – learn from global thought-leaders, network with hundreds of software testing professionals and celebrate everything that's great about your profession over four intensive days of knowledge sharing in a vibrant atmosphere amongst your testing peers.

The 2015 Programme offers practical advice, real-life experience stories and thought-leading insights into DevOps, Mobile, Management, Communication, Agile, Test Automation and more – presented by a mix of experienced speakers and newcomers to EuroSTAR. Check it out here.

Maximise your savings on conference tickets with a Group Discount.
Exclusive 10% discount for **Tea-time with Testers** readers when you use the code **TWT2015**

TAKE YOUR TEAM TO MAASTRICHT
GET 5 TICKETS PAY FOR 4!

GET TICKETS!

EuroSTAR Software Testing CONFERENCE
Europe's #1Software Testing Conference
Maastricht 2-5 Oct

TEA-TIME WITH TESTERS

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

Created and Published by:

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

Editorial and Advertising Enquiries:

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

Lalit: (+91) 8275562299
Pratik: (+49)15215673149

Editorial

The power of zero

While I was out for some shopping with my wife in a mall, I happened to meet a gentleman there. He came near me and asked if I was editor of Tea-time with Testers magazine. Of course I felt good about it and offered him a coffee ☺.

He told me that he was Dev + Test manager of his team and has been reading Tea-time with Testers with interest (wonderful). But then he raised the concern that he found his test team (which was full of fresh testers) very weak and was thinking to scrap it all together. He did not even bother to call them Zero against his Dev team of Heroes.

Naturally I felt bad about it but I avoided to comment on it since I did not know enough context. He asked me what he could do about it and I told him below story.

0 1 2 3 4 5 6 7 8 9

One day, number 9 slapped number 8. "I am bigger than you hence I slapped you", number 9 said to 8. Following the same, number 8 slapped the 7 who in turn slapped number 6.

This continued till number 1 was slapped. But number 1 was smart. Know what she did? Instead of slapping number 0, she asked 0 to come her side and joined hands with her. Number 1 thus became 10. The most powerful and biggest of them all.

The point is, there would be many such (so called) zeros around us. If we think, we are powerful than them then we should help them become stronger rather than bullying them or alienating them. We should not forget that such zeros have tremendous potential to make us even more powerful. All we need to do is, recognizing their potential and empowering them so as to become more powerful ourselves.

The gentleman then said he needed to leave but promised me to meet again with story of "how he made 10 out of 1 and 0". Guess he was the one. I hope he succeeds and also hope that 2,3,4,5,6,7,8,9s in our field do the same.

Sincerely Yours,

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

QuickLook

On-Demand Webinar: Transform a Manual Testing Process to Incorporate Test Automation

Although most testing organizations have some automation, it's usually a subset of their overall testing efforts. Typically the processes have been previously defined, and the automation team must adapt accordingly. The major issue is that test automation work and deliverables do not always fit into a defined manual testing process.

Jim Trentadue explores what test automation professionals must do to be successful. These include understanding development standards for objects, structuring tests for modularity, and eliminating manual efforts.

Jim covers the following in this session:

• The revisions required to a V-model testing process to fuse in the test automation work.

• How to do a gap analysis for those actively doing automation, connecting better with the functional testing team.

Learn how to transform your manual testing procedures and how to incorporate test automation into your overall testing process.

Jim Trentadue has more than fifteen years of experience as a coordinator/manager in the software testing field. As a speaker, Jim has presented at numerous industry conferences, chapter meetings, and at the University of South Florida's software testing class.

Brought to you by - Ranorex®

TESTBASH GOES TO AMERICA!

TESTBASH NYC

On **November 5-6th 2015** we will gather to hear, learn from, meet and speak to other people who love software testing.

TestBash is a conference dedicated to the art of software testing. We are an open and friendly community of people who are dedicated to improve the world of software testing. We believe change will happen through learning opportunities and through making friends.

We are a friendly bunch. Come join us. You'll soon feel at home. **Know more** about the conference…

Hear directly from people what they think about TestBash.

TESTEA TALKS

Evolution of the Art and Craft of Testing cannot be discussed without citing credits to the work done by **Dr. Cem Kaner** in last few decades. Dr. Cem's contribution to this field has had a significant influence on the way testing is being done by thinking and adaptive testers today.

Conversing about testing with Dr. Cem, knowing more about his journey and knowing more about his opinions on various subjects in the field was on our wish list from long time. I got to speak with Dr. Cem on various aspects of his life, expertise, opinion and suggestions recently. And this 'series' is a result of what we have been discussing from quite some time.

In part 2 of interview with Dr. Cem Kaner we mainly discussed how Depth and Diversity can help tester improve their prospects in software testing. In part 1 of the interview, you read about his career journey with some interesting stories and lessons learned. We also discussed about his views on getting testing education from commercial training courses and from universities.

To catch up, you can read part 1 here and part 2 here. Read on and find what invaluable insights we get from him this time....

Lalit Bhamare

Over a Cup of Tea with Dr.Cem Kaner

You have said that testers are at risk of becoming irrelevant. What is your concern?

Many of our favorite methods for designing and running tests are the same as they were in the 1970's and 1980's. I think that many of our discussions of testing process are also stuck in the 1980's. We're still complaining about the same conflicts and trying to deal with them in, largely, the same ways.

The problem is that the world around us has changed. We used to call a 10,000-line program "big". These days, many programs have millions of lines of code. Programmers work in new languages, with better development environments and much richer libraries. The libraries let them add much more capability—from a testers' point of view, much more complexity—much more quickly.

If your testing approach has not gotten much more efficient, your testing staff has not gotten much larger, but the software you are testing is more than 100 times longer and more complex, then your testing has to have a smaller impact than it used to have. The more quickly that programmer capability rises, compared to tester capability, the less relevant the testing is to the final product.

Have we learned so little about how to test better over the last 30 years?

I think we've learned an enormous amount about how to test. I think we understand a lot more about the ways that programs fail. We've developed many new techniques for predicting where errors will be, for noticing errors more quickly, for verifying that revised code has not changed in unexpected ways, for tracing changes in data, for creating, executing and monitoring the results of tests and for finding patterns in failures.

The problem is not in the state of the art. It is in the state of the practice.

What is the difference between state of the art and state of the practice?

The "state of the art" is about the leading edge of the field. What do the most knowledgeable people know? What can the most skilled people do?

The "state of the practice" is about the normal practice in the field. What do most testers know and do?

I think the state of the art in testing has evolved enormously over the past 30 years. I think the state of the practice has barely budged. We keep running around the same circle and telling ourselves we are making changes, we are moving, therefore we must be making progress. We invent new names for ideas that were popular 15 years ago, then sell them as new ideas.

... CONTINUED ON PAGE 55

Tea & Testing with Jerry Weinberg

The Courage Stick

Fear is the main source of superstition, and one of the main sources of cruelty. To conquer fear is the beginning of wisdom. —Bertrand Russell

The Courage Stick is a metaphor that reminds us of that we have the courage to try new things, and to risk failure. Without my courage stick, my consulting turns to grape jelly—all those things I've refined by doing them over and over, too scared to try something new.

The Coward's Credo

The first problem I'm having with this article is that, unlike other topics in my More Secrets of Consulting book, I have no direct experience of courage. In reality, I've never done a courageous thing in my life.

My dictionary says that courage is mental or moral strength to venture, persevere, and withstand danger, fear, or difficulty. I have overcome a number of difficulties in my life, by that's not courage—it's stubbornness. But danger is something else. Tom Crum defines FEAR as an acronym. One translation is "Fantasy Experienced As Reality," and we'll get back to that. But his definition that fits my behavior is "F--- Everything And Run." I'm a total coward, though I suppose many people don't perceive me that way, so let me give a couple of examples of what may appear to take courage:

- Admitting to my readers I don't know much about this subject.
- Writing this chapter anyway.

No matter how much these two acts look like courage to you, they don't feel like courage to me. Why not?

Consider admitting that I don't know much about this subject. I suppose that many consultants seem afraid of confessing ignorance on any subject, but that just doesn't bother me. What I fear is being ignorant and getting caught pretending I'm not. That would present a true danger to my success as a consultant, so, if courage is persevering against danger, the courageous thing to do would be to pretend I know all about courage —and risk being found out.

Similarly, it seems to take courage to start writing a chapter when I don't know anything about courage. But that's my Golden Key at work—I write this so I can learn about courage. What I really fear is not understanding the subject—so the courageous act would be to abandon the writing altogether, rather than succumbing to my fear of ignorance.

But in spite of my reasoning, my readers continue to tell me how "courageous" I am to write about certain subjects, or to write at all. Over the years, this sort of balderdash has brought me to believe in the Coward's Credo:

Courage is not a feeling, but an outer appearance.

I don't mean to say that there aren't people who feel courageous, but just that I'm not one of them, and I don't understand them at all. Any courage of mine you see is just an illusion, and what lies beneath that illusion could be one of several things:

- A different perception of the risks and rewards of an act
- A different knowledge of the facts surrounding the act.

How To Use Your Courage Stick

I tried to research courage among graduates of our Change Shop, where we teach the use of the Courage Stick. I didn't get very far, because "courageous" people told me, as I told you, "I wasn't courageous." So then I asked "What do you do, actually, when you use their Courage Stick?" Here are some things they said:

- "When I found myself being afraid of how hard it would be, I used the image of my Courage Stick to remind myself of things I'd done in the past that were harder. Like, I once had four wisdom teeth extracted without anesthetic. That made this task seem trivial, in comparison."

- "I was afraid it would come out badly, so I wasn't going to do anything. I touched my little Courage Stick on my charm bracelet just to remind myself to stay calm, and then I started thinking of other outcomes besides the one I feared. I saw that there were many possibilities, not just the first and most frightening one that came to my mind, and I realized that this one wasn't really likely, but only scary."

- "I knew that there were other possible outcomes, but at first I didn't think they were very likely. When I thought of my courage stick, I imagined myself walking into the situation with some power to act on my own behalf. Soon I was thinking of other factors that could influence the outcome, factors that I could easily influence."

So what, in fact, does the Courage Stick do? In effect, it reminds us of another meaning of the acronym, FEAR:

FEAR = Find Every Available Resource

Once reminded, I shift from my paralysis to a search mode, finding other resources. I then explore the facts thoroughly and consider the risks and rewards carefully, not reacting to the first fact or risk that comes into my mind. And, if the risks and rewards are still unfavorable, I think of ways I can act to alter the outcomes. Having thought of all these things, I may then act in a way that looks courageous to the outside, but is actually the result of calm, cautious contemplation.

Sometimes people want to choose a representation of their courage that's a large bludgeon, like a baseball bat. This is a mistake, based on the concept that courage comes from the ability to beat up other people.

Sometimes people want to choose a representation of their courage that's a tiny pointed thing, like a needle. This is a mistake, too, based on the concept that courage comes from the ability to needle other people—to manipulate them.

The Courage Stick represents the calm and comfort needed to think effectively, and the ability to handle ourselves well, regardless of what other people do. A proper Courage

Stick can be held calmly and comfortably in one hand, and has no sharp points or edges. My favorite Courage Stick is a piece of wood I found on the beach in Oregon. It feels as if it were made for my hand, smoothed by countless encounters with salt waves and sand. I reminds me of how many rough encounters I have survived, and that gives me the calm to appear courageous.

Albert Einstein once said, "Great spirits have always found violent opposition from mediocrities. The latter cannot understand it when a man does not thoughtlessly submit to hereditary prejudices but honestly and courageously uses his intelligence." That's what the Courage Stick can do—remind you to honestly use your intelligence, rather than simply reacting to the fears from your first prejudgment of the situation.

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.**

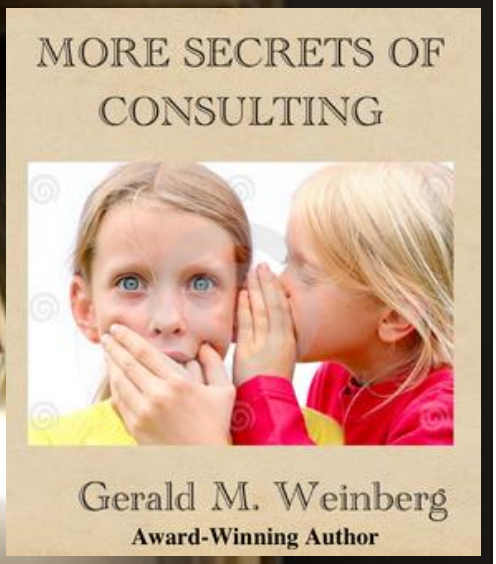
To know more about Gerald and his work, please visit his Official Website here .

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

Widely acclaimed as a consultant's consultant, Gerald M. Weinberg builds on his perennial best-seller The Secrets of Consulting with all-new laws, rules, and principles. You'll learn how to fight burnout, stay curious, understand your clients, negotiate effectively, and much, much more.

Consultants need more than technical skills—they need self-awareness and a strong set of personal abilities. Weinberg helps computer consultants identify and strengthen each aspect of their performance using a "consultant's tool kit" of seventeen memorable symbols. He devotes a chapter to each of these symbolic tools, from The Wisdom Box to The Fish-Eye Lens to The Oxygen Mask. If you aspire to be successful consultant, this book is for you.

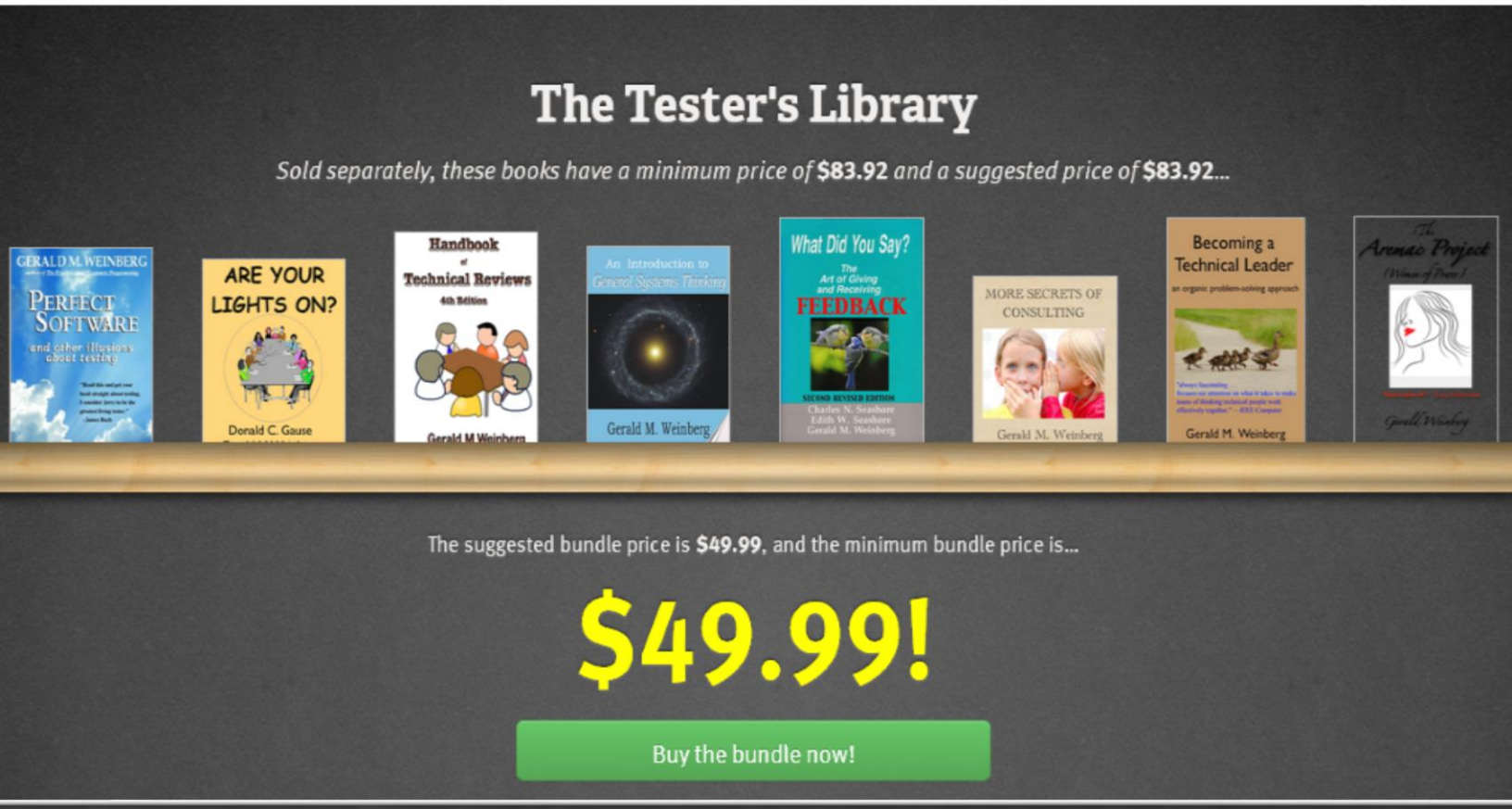
Know more about Jerry's writing on software on his website.



TTWT Rating: ★★★★★

The Bundle of Bliss

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



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

The 8 books are as follows:

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

Know more about this bundle


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

I feel, software testing is challenging and it's even more challenging for fresh testers.

In my opinion, new testers often feel that regardless of how much effort they spend on researching software testing and improving, they still don't feel that they are making any progress or worse they are not sure if testing is the right career for them.

If you are a fresh tester and feel the same, I think it's not because you are incapable, it's because you are worrying too much about things that do not really matter much.

I feel, by stopping worrying these things, you will be fine and would be able to focus more on important things that matter.

1. Terms and Definitions

This is the number one worry I believe even though there's no official statistics about this.

Why I believe so?

In most of the software testing forums, online discussion, the following questions have been asked:

What is QA, QC, Tester?
What is the difference between QA, QC, and Tester?
What is the difference between Bugs, Defects, Errors, and Faults?
What is the difference between Test Plan and Test Strategy?

I am not saying that you should not ask these types of questions. When you don't know a term or are confused, it's natural for you to ask these questions. However, if you ask these questions again and again with intent to find the BEST definition, you are wasting your time because these types of questions often lead you to nowhere.

You know the beauty of terms and definitions? Well, people can define things that suit best to their context. Unless you want to be a collector of terms and definition in software testing, I suggest you to stop worrying too much about terms and definitions. Pick the one that best suits your context and go ahead with it.

2. Rejected Bugs

Let me guess. You spent several hours exploring the system and found an amazing bug (at least you think so). You happily reported the bug on to Bug Tracking System and a few hours later, the bug was rejected by developers or your manager.

Note: by "rejected" I mean the bug is invalid and you agree too. You feel like you are "Rejected" and then you worry "I had several rejected bugs this month, I wonder if this affects my testing productivity?"

"Developers are laughing in my face for reporting an invalid bug"

"My value is going down"

"Developers don't like me"

"I have a problem with my testing capability"

Take it easy friends.

When you test a product, you know something about the system under test, but there are also things that you don't know yet. Testing is a learning process. When you test, you go from the known things to the unknown things.

If you have rejected bugs, it could be because you are not updated on the changed features or even don't know if the feature exists. Learn from it and move forward. What I'm trying to convey is that don't let rejected bugs discourage you from finding more good bugs.

Side note to managers: if you are judging tester's productivity based on counting bugs, can you please just stop?

3. "I missed a bug"

This may be one of the scariest phrase testers don't want to hear from their boss. This is one of the most common misperceptions in software testing.

You feel guilty and worry for every bug escaped and found in the field because:

- Your boss is counting the number of bugs and basing your productivity based on that
- You consider yourself as Goalkeeper who catches all bugs in the system. If one single bug is missed, you think you are not doing the job right.

If those are the cases, this is a good reason why Blame Game starts or the so-called CYA.

"No, it's not my fault, the bug is missed because [someone] did [something]"

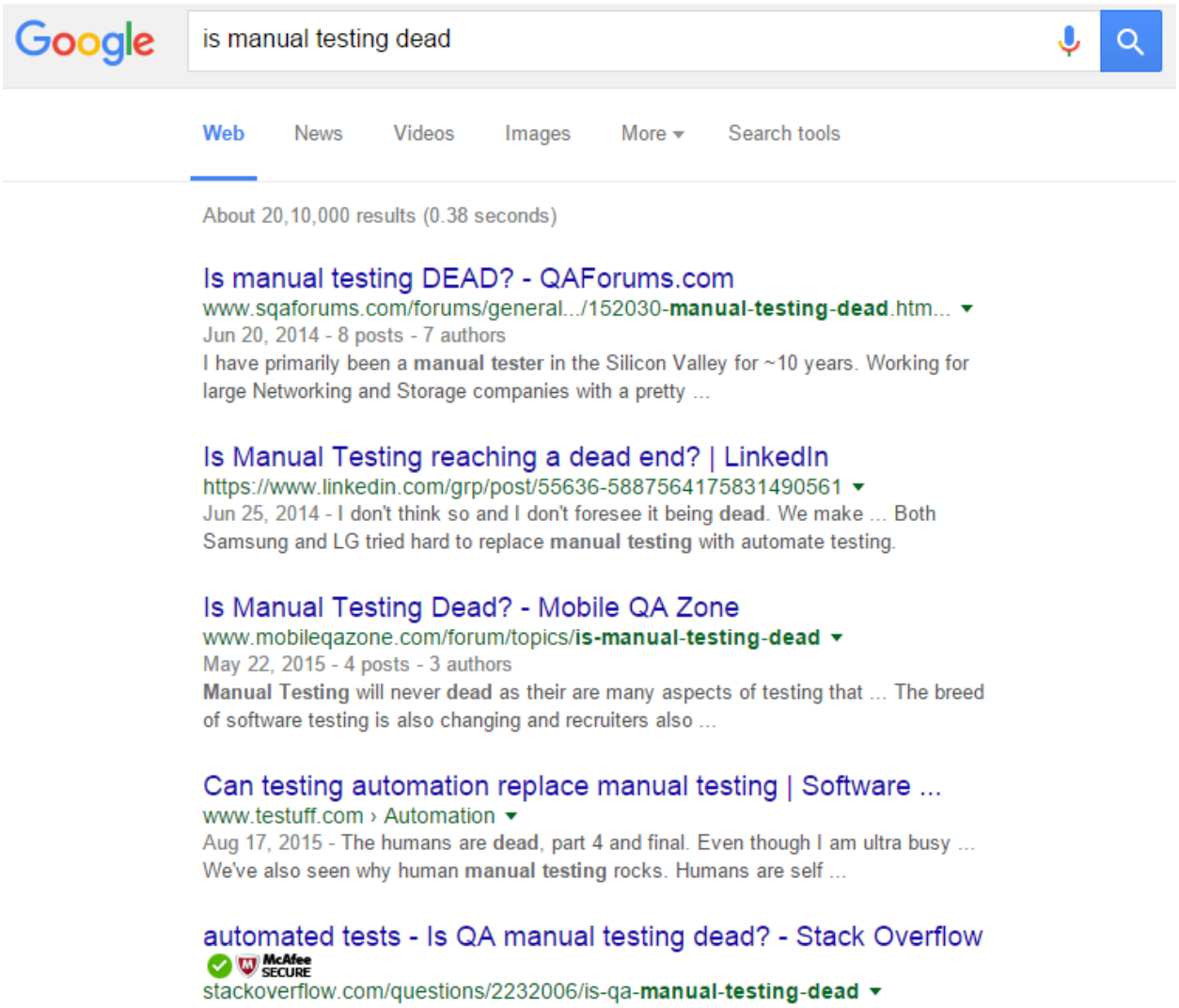
Or you will report every single bug without really bothering about its quality and insist developers to fix them all before releasing the build.

Instead of worrying too much about bugs missed, you can work with team and analyze why you missed it. This is not a process to find who is responsible for the problem. This is a process to see how we can avoid the problem from happening next time.

4. "I'm just a manual tester"

With the appearance of automated test, manual testers feel like they are being threatened and replaced by automation tools.

"Who needs manual testers while there are powerful tools out there can run thousands of test cases with an eye-blink?" Below is a search result for the term "is manual testing dead"?



It looks scary!!! Looks like manual testing is dying or at least testers are worrying about that.

Recently, I did a short interview with Augusto (you can read the entire interview here) and here is what Augusto said about this

"I think that manual and automated testing is a false dichotomy. They are two completely different solutions that resolve two completely different problems; the only link between them is in the name. Our industry, in particular, some tools vendors, have made a mess by conflating the two concepts. This has confused testers and also encouraged companies to use the wrong tools for the job. I believe that we need both, test automation and exploratory/manual testing, as I said before, they resolve different problems."

If you need more to convince you, check out these two blogs from Michael Bolton

http://www.developsense.com/blog/2013/02/manual-and-automated-testing
http://www.developsense.com/blog/2009/08/testing-vs-checking

Sooo….no, manual testing is not dead until automated tests can help ask great questions to challenge the system under test.

5. "I don't have any documents, how can I test?"

If you are a tester transitioning from a traditional set-up to Agile shop or you are in half-baked project, you may find it shocking when you have to test without a handful of specification documents

You don't know where to start.
You have no clue of sign-off requirements.
You don't know how to baseline your testing.
You are totally confused and get lost.

Don't panic. Having no documentation to test is not the end of the world.

Instead of worrying and depressing yourself, try the following:

• If you don't see documentation, just ask.
• Talk with anyone you think can help you understand the system and have them walk you through the system under test.
• Perform exploratory testing and following oracle heuristics (FEW HICCUPPS). I recommend "Testing without a Map" by Michael Bolton for the guidance of this kind testing. **Click here to download "Testing Without a Map" paper**

6. "How to get a first job in software testing?"

To some extent, finding your first job in testing is the biggest worry ever. This is especially true if you are a fresh graduate or you are moving to software testing from another industry.

It appears that:

• There are hundred thousands of experts out there with years of software testing experience
• Your testing experience is zero
• Most of the job ads require at least 1-2 years of experience in testing… for junior tester position

Let's skip crazy job ads like these from recruiters who have no idea what they are talking about in their job ads. What I can say is that no, you are no in a dead end. There are ways out.

If you don't have experience, build it.

There are many organizations out there where you can join projects to find bugs. The great thing is that you are paid and can work from home. I recommend uTest for this type of project.

How about testing on real products?

You can also practice testing on real products like Facebook, LinkedIn and even on my site AskTester. You may add the following into your CV something like "I found some bugs on Facebook /LinkedIn and one of them crashes the system. Here is what I did…"

The idea is to show people how you really do the testing.

Have I told you that joining forums and online discussion helps too?
Join forums and online discussion to ask questions. The more you ask, the more you learn.

Joining the online discussion will not only help you learn new things in testing, but also make you noticed in the software testing community. Recruiters may find your great questions and passion in software testing on online discussion forum such as LinkedIn and may contact you directly for the job. Yeah, who knows?

7. "I fear Changes"

People fear change. If not all, at-least most of them. That's the fact, I believe.

As a manual tester, you worry when you are assigned to automation project where you have to write automated script.

As a traditional tester, you worry when your team is now transforming to Agile where someone one says that you have to test without any document and it's a chaos.

As a tester, you worry when your project progress does not follow initial plan and your testing schedule is affected badly.

I can list more, but you get the idea right?

You resist to change because you worry of uncertainty. You have no idea of what's waiting for you ahead and if you are competent enough to deal with changes. What I can tell is that there's nothing for you to worry about in those situations.

If you worry about automated test because you don't know how to code, pick up a programming language and check out some courses online to learn. No, you won't become an expert after learning those courses, but it will provide you enough basic knowledge to start off

Is Agile scaring you off?

Testing in Agile is still testing. The core work is testing. The only difference is the way you approach and do the testing. You can research to understand more about Agile (see Agile manifesto)

So, instead of worrying and resisting to change, let's do a bold action:

"Let's welcome the changes"

"Change is hard because people overestimate the value of what they have-and underestimate the value of what they may gain by giving that up." - James Belasco and Ralph Stayer

Final thoughts

I've only listed out 7 worries that I feel most of testers have to deal with and I had to deal with as well. The earlier you stop worrying about those things the better you will be doing. Most of the worries I mentioned above turned out not to be big problems at all. It took a few years for me to realize that I worried the wrong things in the wrong way and I wished I could realise sooner. Now if you are a tester and you are worrying about these things, stop worrying right now and do something about them… and trust me, you'll be fine.

Back To Index

Thanh Huynh is owner of AskTester (http://www.asktester.com) and also a tester. He cares about how to do better testing and how to help other testers do better testing too. That's why he built this site.

Thanh has several years of experience in testing, managing and leading testing projects. Beside of 9-5 job, he is also interested in other testing related activities such as technical review, blogging, conference speaking and Tester community building.

If you don't know Thanh, he is kind of introvert, quiet and family-oriented. When he is not doing testing, he spends time on reading blogs, books, playing with his little boy, drinking coffee and listen melodic metal...well, sometimes at the same time.

Love to Write?

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

Send your articles to editor@teatimewithtesters.com


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

Applying Exploratory Testing in a Scripted Testing Environment

- by Jan Jaap & Miranda Kerpel

The popularity of exploratory testing is increasing. More organizations see it as a serious and in some situations preferable way of testing and the number of testers who apply exploratory testing is growing.

But still there are many organizations who only use scripted testing. This article describes a case of an organization that chooses exploratory testing for a project instead of the detailed scripting method. What were the reasons for the change of test method and how did they experience exploratory testing?

The context

The organization is a facility services provider. The program is to make a de-centralized organization with many part-time and temporary employees more centralized. IT is essential for HRM processes and planning processes. The IT organization consists of approximately 50 persons. All projects in the program were using detailed scripted testing. A typical scripted testing process was applied: product risk assessment, test plan, test preparation, test execution, reporting and closure. This was done in a satisfying way for both testers and management, there was no urgent need to change the testing process. Until one project came along that seemed to need another way of testing, because the way of development was more agile than the previously used waterfall methodology. In accordance with the project team the tester decided to test using exploratory testing.

The project in which exploratory testing was used was the implementation of new legislation concerning temporary employment contracts. This organization works a lot with temporary employment contracts and has a high employee turnover. Because of this the consequences of the new laws are big, so the organization thought of an automated control system to manage all temporary employment contracts. A major problem for the project team was that the details of the law where not clear until shortly before the implementation of the law and the implementation date was fixed. It was a new application so the team couldn´t use old test scripts. The project team was pretty small, 5 people including one tester. This tester was experienced in scripted testing and the processes and systems used in this organization.

Based on the situation, the tester came to the conclusion that the usual way of testing was not suitable for this project. In general there was an idea what the law would look like but this information was not specific enough to be used for detailed test scripts. Based on the expected timelines the information needed for detailed scripting would be available too late. The tester, a contractor working for a consultancy firm, was introduced to exploratory testing and session based testing by her employer and she had some experience with it. In various previous projects she used it besides detailed scripting. Based on the situation she decided to propose the use of exploratory testing in this project. In general there were doubts about the desirability and applicability of exploratory testing and session based testing but the project team saw the need to change the way of testing. Another reason to change the way of testing was the application of Agile in this project. The tester was convinced exploratory testing or session based testing fits better in the Agile environment and explained the method of testing to the project team. They came to agreement that they should use this way of testing.

Structuring exploratory testing

Exploratory testing is by some confused with unstructured testing, ad-hoc testing or unprepared testing. Exploratory testing is an approach to software testing that is described as simultaneous learning, test design, test execution and analysis. Applying exploratory testing doesn't mean there is no structure or preparation whatsoever. For an explanation of exploratory testing see **'Explore it!' by Elisabeth Hendrickson** (The Pragmatic Programmers, 2013). In this project every situation recognized in the concept law was a test unit, the importance of the situation to the organization determined the priority for testing. The recognition and prioritization of the situations was done in a session with different stakeholders, this session could be seen as a test strategy session. The participation of different stakeholders increased the involvement of these stakeholders in testing.

For every situation a test charter with one or more goals and some test ideas was created. Later on in the project test ideas were added if necessary, especially during and after test execution. When the charter was executed all test ideas should be covered. There were no detailed test scripts or something like that made upfront.

Before the system was delivered, the tester had no insight in the system at all. She studied the concept law and spoke with subject matter experts about the situations and the details about how the system should work but no other preparation was done upfront. The testing itself was really exploratory in the sense that the tester knew the goal and the test ideas of the test session but was free to decide how to test and adjusted her testing based on what happened during the test. During testing notes were taken to be able to show what had been tested and to determine the test coverage. The notes included test actions, outcomes, obstacles and defects. The defects were also captured in a defect management tool, the handling of defects in exploratory testing was not different compared to scripted testing. During testing a big matrix with the situations to test was used to measure coverage and progress. This matrix was visible for everybody in the team so the coverage and progress was clear for everybody. The outcome of the tests was used to adjust the priorities, the priorities were discussed frequently in the team, especially with the subject matter experts. During test execution, the regression test set was built. After testing a situation the tester decided which test cases should be part of the regression test set and if possible these test cases were automated immediately. This way the regression test could be executed easily and this increased the confidence in the system by the business.

Exploratory testing considered

In this article one project in which exploratory testing is used is described. Though this may not be sufficient to judge the usefulness of exploratory testing in general, we can discuss how this organization and tester experienced the use of exploratory testing.

Upfront, the organization was very critical about exploratory testing. The old way of working with making detailed scripts upfront worked pretty well according to most people involved. Letting go of the old way of working, especially the lack of previously made detailed test scripts, made some people doubt. They just couldn't imagine how test execution could be done without the guidance of scripts, they were scared that the quality of the test would decrease due to the lack of preparation. Upfront, most people involved couldn't get the idea of simultaneous learning, test design, test execution and analysis. It took some time and persuasiveness to apply exploratory testing in this project. In the end they needed to trust the tester, who had a very good track record in this organization. The experience of the tester and some experience reports of other projects in other organizations were necessary to convince these persons.

Once started the tester (and other team members) didn't feel really comfortable, the absence of test scripts gave the tester a sense of uncertainty. During testing the tester found out she could respond more easily to new information and new (or changed) risks. Due to the lack of information upfront this flexibility was essential to test this system in the given timeframe.

During test execution the tester found out flexibility introduced a risk: the risk of undisciplined testing. Without the guidance of test scripts a tester can easily get caught in the pitfall of unfocussed, undisciplined testing. The use of focus/defocus helped the tester to avoid that; she was sometimes completely focused on the details of her testing and sometimes (at least once a day) de-focused to judge whether she did the right thing and did everything right. She discussed this regularly with other team members and changed priorities. Nevertheless discipline remained a major concern.

One of the biggest advantages of exploratory testing in this context was the flexibility for both the tester and the team. New information could be integrated in the test approach on the spot. The outcome of a test was used as input for new tests. And although it is not supported by measurements, everybody is convinced that even though less time was spent on testing relatively more time was spent on actual test execution.

Looking back, everybody involved in this project was convinced that exploratory testing was the best way to do testing in this project. But to be applicable some conditions must be met. First the tester must have good knowledge of the business processes. The tester had time available to study the situations, the processes and the concept law upfront. By the time she started testing she had good knowledge of the business processes. Second, there should be short communication lines and commitment by the team and the business. Business representatives must have time available to support the tester(s) and team members and business representatives must be willing to let go of the (false) sense of certainty given by detailed test scripts. All people involved must be confident the tester(s) is (are) doing the right things. Short communication lines are essential for this.

And of course the application under test and the project characteristics must be right for exploratory testing. A system with many complex, detailed calculations? It's probably better to use detailed scripting. A system with very much user interaction? Then a form of exploratory testing is most likely better. So exploratory testing is not better compared to scripted testing, it depends on the characteristics of the organization, project and system.

Introducing exploratory testing in an organization that feels comfortable with detailed scripting isn't an easy step. Detailed scripting gives a, sometimes false sense of certainty and people with a lot of experience with detailed scripting sometimes just can't get their head around a concept like exploratory testing. You need a specific project for a system that differs from the other projects and you need a tester with experience with exploratory testing and persuasiveness.

So choose your project and tester carefully when you want to implement exploratory testing. Ultimately, in some situations, exploratory testing is better, faster and cheaper compared to detailed scripting. Mature test organizations and professional testers should be able to apply both.

Jan Jaap Cannegieter has over 20 years of experience in testing, quality assurance, requirements, TMMi, CMMI, SPI and Agile. Jan Jaap is Vice President of SYSQA B.V., a company with 180 employees specialized is requirement, quality assurance, testing, IT-governance and process improvement.

Jan Jaap is a well-known author of several books and articles and is regularly (keynote) speaker on international conferences. He is one of the authors of 'Situational Testing', see http://www.sysqa.nl/publicatie/ebook-situational-testing-english/

Miranda Kerpel is senior QA professional of SYSQA B.V, specialized in quality assurance, test automation, test coordination, requirements and testing. Miranda did several assignments as information analyst, tester, test manager and quality manager at different organizations.

She is chairwoman of the knowledge group test automation within SYSQA B.V

Abstract

During the development of a data warehouse, huge amounts of data are transformed, integrated, structured, cleansed, and grouped into a single configuration that becomes the data warehouse. With this variety of source data, and often complex types of changes, data corruption may result. Therefore, DW testing is a critical stage in the DW development process.

Many attempts have been made to describe how the testing process should be accomplished in the DW environment. In an earlier issue of Tea Time for Testers, three important challenges of data warehouse testing were described:

- Identifying the specific focus areas of data warehouse testing
- Identifying project documentation that's needed for data warehouse test planning
- Choosing qualified testers for data warehouse QA

In this this article, three more testing challenges are described and how each challenge can be met.

Testing the data warehouse system outputs in comparison with the data in the sources is not the best approach to identify alone, the quality of the data. However, this type of test is always important and will take place at a certain point in the testing process. Every stage and every component the data passes through should be tested to guaranty its efficiency and data quality preservation or even improvement (see Figure 1).

Primary intentions of this article are to:

1) Identify common data warehouse testing challenges and
2) Highlight approaches and guidelines to address those challenges.

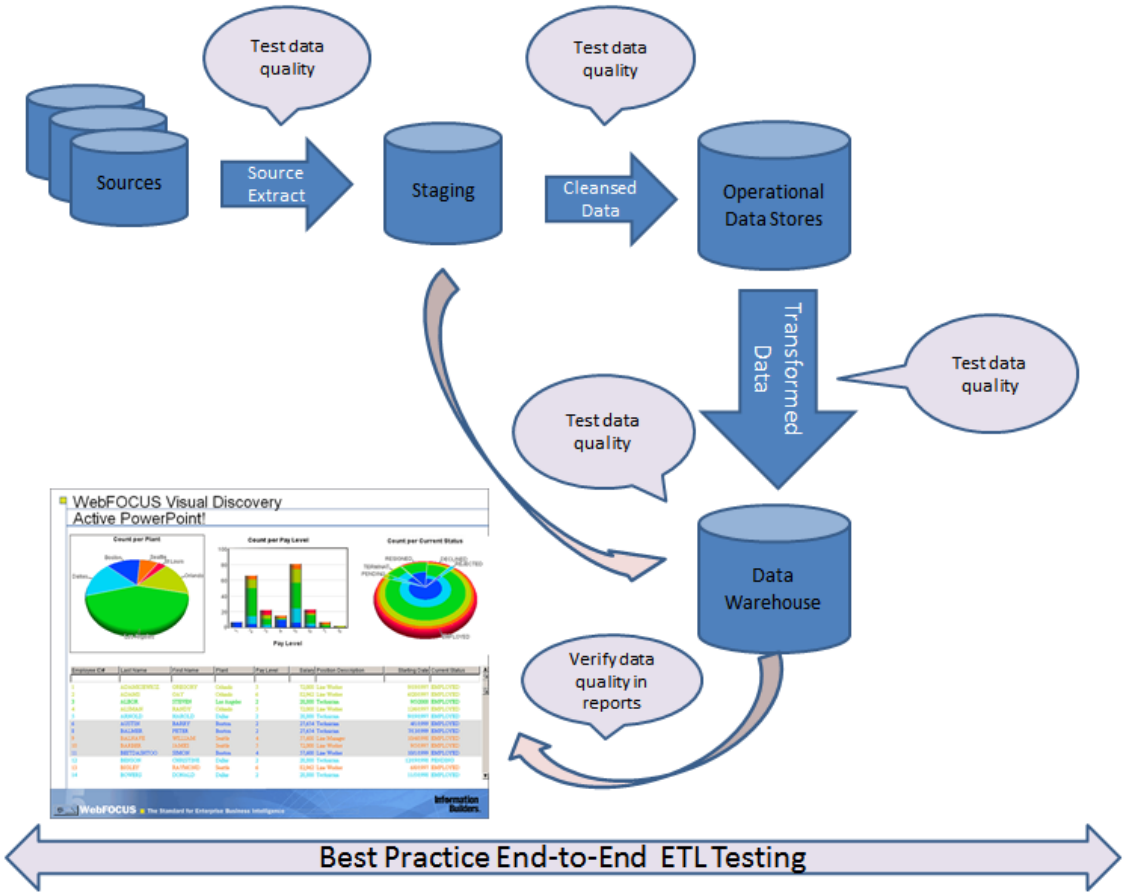


Figure 1: Testing the data warehouse – the many phases where testing is critical

Introduction

Data warehouses are becoming increasingly large, increasingly complex and increasingly important to the businesses that implement them. They are becoming increasingly large and complex because they are drawing more data from a greater number of more diverse sources across the enterprise to create larger, richer assemblages of both alphanumeric and financial information. They are becoming increasingly important to the business because they are being leveraged by a wider range of users to support a greater number of decisions that impact the bottom line every day.

For those reasons, it's essential that businesses rigorously ensure the quality of the data in their data warehouses. If they fail to do this, users will make faulty decisions based on incorrect data. Over time, their confidence in the data will erode to the point where they won't use the business intelligence tools and other applications that rely on the data warehouse – which mean huge investments in IT will be wasted. Just as important, any business using financial data in its data warehouse must be able to withstand the scrutiny of auditors and regulators.

In other words, without effective data quality management measures in place to support their data warehouses, businesses will remain highly vulnerable operational, financial and regulatory risks. Unfortunately, few companies have adequate safeguards in place for their data warehouses today. They

may have conventional tools in place for validating certain types of data (such as customer names and addresses) once they're in the data warehouse, but they often lack the controls necessary for…

- Preventing bad data from getting there in the first place
- Properly validating financial data
- Discovering and remediating the root-causes of chronic data quality problems
- Documenting data quality management measures to third parties – such as auditors and regulators

Test planning for a DW/BI project should be designed to overcome the most significant challenges faced by data warehouse testers. Three of those challenges are described below.

Challenge 1: Identifying and correcting source data quality defects

The initial load of a data warehouse is often a very time-consuming process. It's important to assure that there is substantial comfort with the quality of data being loaded into the warehouse before investing several days, in some cases weeks, executing ETL processes.

A leading cause of data warehousing and business intelligence project failures is identifying, then loading, the wrong data or poor quality data. The source system consists of all those "transaction / production" raw data from where the details are pulled to make it suitable for the data warehouse. All these data sources usually have their own methods of storing data. Because of this diversity, several conditions may be present which could contribute to data quality problems if proper care is not taken.

Most data sources are legacy systems or vendor supplied sources. These systems are usually managed by varied organizations in different business departments. Since the business lines supported by these systems are different, users of one system are often unaware to the features or capacities of the other system. For this reason, many business processes and data are duplicated across systems and the semantics are diverse. For example, the definition and calculation of revenue in the "wholesale" sector may be different from that of "retail sales" sector. Or the list of customers maintained in the "sales" organization may be different in quantity and metadata quality with the list of customers maintained in "marketing" department.

Different data sources have different kinds of problems associated with them such as data from legacy sources (e.g., mainframe-based COBOL programs) not having metadata that describes them. Other sources of unclean data include data entry or update errors by a human or computer system. And, some of the data may originate from text files, Excel files, and some may be directly from ODBC connections to the source database.

The data warehouse environment may be unique in that it is the source of information used by the business to make strategic decisions, but it does not actually create much data. That means that data quality problems in the data warehouse often originate in source systems, are created because of faulty data acquisition and delivery processes, or are due to interpretation and transformation glitches when loading the data warehouse.

Data quality problems in the source systems need to be recognized, and the data warehouse team is responsible to either address these problems or gain business concurrence that the problems are acceptable. The data warehouse team must then ensure that the data warehouse users are aware of these data quality deficiencies.

Faulty data acquisition and delivery processes are fully within the control of the data warehouse team, and that team is responsible to ensure that these don't happen. Data interpretation problems also need to be addressed by the data warehouse team. The responsibilities of the business analyst and data analyst include establishing clear definitions for each data element and ensuring that the data acquisition and delivery specifications instruct the programmers on how to properly populate the data element.

Table 1 summarizes possible causes of data quality difficulties in the data sourcing stage of data warehousing.

According to a Standish Group Report, one of the primary causes of data migration project overruns and failures is a lack of complete assessment of source data prior to data movement to the data warehouse. Each of the source origins presented in Table 1 are related to the data sources which are the feeder systems for the data warehouse. One of the fundamental obstacles in current data warehousing environments are concerns with the existence of inconsistent data.

	CHECKLIST FOR SOURCE DATA QUALITY PROBLEMS	**RISK MITIGATION EFFORTS**
1	Check for inefficient data source selection and profiling for candidate data warehouse data may cause data quality problems (i.e., source data that does not comply to business rules). Examples: • An email column does not always contain valid email addresses • Birth date field has values older than 150 years • Combinations of customer name and address columns are not unique	For each data source attribute / field, run SQL queries or a profiling tool to check the data for compliance to business rules. Verify that the data in each field complies with business rules and business needs. Recommend maintenance enhancements to data acquisition processes to improve accuracy of data warehouse data.
2	Check for insufficient knowledge among source selection team of inter-dependencies among data sources used to populate the data warehouse.	Review the quality of data entry, loads, transformations, extractions, merges, or other jobs utilized to create each source data file / table. This may also include the establishment and monitoring of service level agreements, communication protocols with data suppliers, and data quality assurance policies.
3	Verify data and time ranges plus timeliness of data sources	The QA team can check whether any of the source data tables are not complete when where relationships are required with other source tables planned as input to the data warehouse, (ex., one source has data since June 2009; a second related source contains no data for that year.)
4	Identify unexpected changes in source systems over time which can affect the history of data to be loaded. The complexity of a data warehouse increases geometrically with the span of time of data to be fed into it. Non-Compliance of data in data sources with the business standards.	Where possible, continually monitor source systems using profiling methods, to determine if unannounced or unexpected changes in data fields are occurring. Review historical integrity of data warehouse source data. Since data warehouse input data is often loaded from internal and external sources, and data often covers a significant time span, application of business rules to that data may be applied to only a portion of the data to be loaded.
5	Check for the presence of duplicate records of same data in multiple sources cause DQ Problems.	Techniques such as comparisons against other data sources, data edit checks, and duplicate record removal can be used to clean source data as it is loaded. While these efforts may yield more accurate, complete, and consistent data in the short term, they seldom result in

		lasting improvement in databases that are experiencing rapid inserting and updating of records.
6	Verify that adequate data quality testing is performed on individual data sources.	Since the level of testing on individual data sources can vary, before data is loaded to the data warehouse, the QA team should confer with owners of source data to learn in detail, their verification process in order to seek out deficiencies.
7	Identify primary key strategies for the same type of attribute (Ex. one table stores customer information using the Social Security Number as the key, another uses the ClientID as the key, and another uses a surrogate key).	Data analysts should consider what each other potential data warehouse source uses as primary key. BA's should recommend maintenance enhancements to data acquisition processes to improve accuracy of data warehouse data. Make recommendations to operational support for enhancements to systems of record to improve accuracy of operation data.
8	Check whether different data types for similar columns are implemented among sources (Ex., a customer ID is stored as a number in one table and as string data in another).	Verify that where data types are different across data sources for similar data (ex., SSN), that any transformations / changes will be applied before loading to the data warehouse)
9	Check for varying default values used for missing data among data sources.	Since default values can change over time for individual attributes, testing should assure an understanding of when default values changed and assure that they are accounted for when loading data in the data warehouse.

Table 1: Checking for source data quality issues

Challenge 2: Gaining project stakeholder participation

The paper trail is a bit like the police. Everyone avoids it until it is needed -- gaining sign-offs on all phases of data and report testing. A stakeholder is a person who has a direct or indirect stake in the data warehouse project. From a project point of view, stakeholders can be categorized by various responsibilities:

- **Accountable** – some stakeholders are accountable for the success of the entire project or a particular phase of the project, generally they are the departmental or divisional managers
- **Responsible** – other stakeholders are responsible for the deliverables, generally they have been charged by management to provide a definitive outcome from the project
- **Consultation** – some stakeholders are used as consultants like an Subject Matter Expert (SME)
- **Informed** – many stakeholders need to be kept informed, generally management who need to know where their investment dollars have ended up or external parties.

Stakeholders do not need to know everything about the technology or even the application set. Working with stakeholders will provide the project with a boost of confidence knowing that the stakeholders are enthusiastic about this project.

In a data warehouse implementation, it is generally the stakeholders who have requested the need for decision support. Therefore, it is assumed that stakeholders will be key supporters of the project as they have most likely been the ones who have identified initial requirements for the data warehouse. Stakeholders require tangible outcomes that they can measure, trust and quantify from the consolidated data warehouse data. It is important to stakeholder(s) as this will provide them with the ability to follow through with the decision making process.

Project team stakeholders will generally consist of:

- The project sponsor – often from the business side
- Steering committee members
- The project manager
- The IT implementation team (business analysts, developers)
- Quality assurance team
- End users

Each member of the project team is, in fact, some type of stakeholder with respective responsibilities and interests in the project. It is crucial to ensure that all stakeholders are identified as early as possible in the project lifecycle and more importantly, to engage with the stakeholders and get their 'buy-in' for the project. Neglecting stakeholders, especially those who have true accountability may lead to continual negativity about the project no matter how highly skilled the project team is or how good the technology platform is.

Stakeholder management also work to ensure a common set of understanding across the project so that success can be measured. A project is in jeopardy when it shows the following traits:

- The scoping of the data warehouse project was ambiguous and no clear sign-offs achieved
- Unrealistic completion timeline was set for the data warehouse and no project plan
- Wrong implementation or a skill-set mismatched in the development and QA team which could lead to incompatibility of technologies used
- Project driven by IT instead of business
- Project manager allowing scope creep to occur
- A data warehouse developed beyond the organization's maturity and capability to appreciate and utilize the wealth of information they obtained in the data warehouse

Without down-playing the roles of other members of the project team, a project without a project sponsor is most likely not going to succeed in the eyes of the client due to lack of ownership at the highest level of the client organization.

Challenge 3: The scarcity of high-value tools for ETL testing and test automation.

Until now, there is no dependable solution available for automating data warehouse testing. Not that it is impossible, but it is very difficult given the lack of standardization in how the metadata are defined and design approaches applied in different data warehousing projects. There are a few commercial solutions that depend on metadata of the data warehouse but they require considerable customization efforts to make them workable. Inaccessibility of automated testing opportunity also implies that the right kind of skill set will be necessary within the testing team to perform such tasks.

ETL test automation challenges

- Large number of tables and records.
- Multiple source systems involved.
- Testing the data synchronization for all these tables.
- Testing Business Objects reports through automation.

Common data warehouse test automation objectives

- Automate as many data warehouse testing cycle activities as possible
- Verify data across all points during the ETL process
- Verify all data, not just a subset or sample
- Verify complex data transformation rules

- Reduce manual testing workload which could be thousands of SQL scripts
- Identify mismatched and missing data
- Profile the performance of the data warehouse
- Focus on numeric financial information for compliance or financial reporting
- Reconcile and balance information between input and transformation processes
- Validate information between the source and its final point in the data warehouse
- Provide assurances to auditors that all financial information residing within a data warehouse can be trusted
- Develop a regression test suite for system testing and production monitoring

Data warehouse processes - subjects for test automation

- File/data loading verification
- Data cleansing and archiving
- Load and scalability testing
- Application migration checks
- Extract, transform, load (ETL) validation and verification testing
- Staging, ODS data validations
- Source data testing and profiling
- Security testing
- Data aggregation processing
- BI report testing
- End-to-end testing
- Incremental load testing
- Fact data loads
- Dimension data loads
- Performance testing
- Regression testing

Deciding which ETL tests should be automated

The trick is to figure out what needs to be automated, and how to approach this task. An array of questions must be considered in the course of automating tests such as:

- How much will it cost to automate tests?
- Who will do the automation?
- What tools should be used?
- How will test results be displayed and what skills must they possess?
- Who will interpret the results?
- What will script maintenance consist of?
- What will happen to the test scripts after the launch of the application?

Not all the areas of data warehouse testing can be automated, but some critical testing involving data synchronization can be achieved using an automated data warehouse solution.

A primary objective of automated data warehouse testing is to cover the most critical part of data warehouse which is synchronization or reconciliation of source and target data.

A path to ETL test automation

A starting point for ETL test automation can be divided into three major areas.

1. *Count Verification*: Initially all the tests for source to target mapping and Incremental Load for verifying initial/incremental/differential data count can be automated.
2. *Data Verification*: Secondly, for source to target mapping and Incremental load data verification tests can be automated.
3. *Data Type Verification*: Lastly, scenarios for checking of the data types of source and target fields can be introduced.

Recommendations for getting started with ETL test automation

Evaluate the current state of testing in your organization. Not all ETL testing requires automation. Assess the situations mentioned above to determine what type of automation would benefit the testing process and how much is needed. Evaluate testing requirements and identify inefficiencies that may be fixed with automated testing. QA teams that spend a lot of time on regression testing will benefit the most.

Create a business case for automated testing. Automated testing generally comes at a high price, so in order to convey the value to the business, the IT team must first make the case.

Evaluate the options. After evaluating the current state and requirements within the IT department, look into which tools fit the organization's testing processes and environments. Options may include vendor, open source, in-house, or a combination of the aforementioned tools.

Automated ETL testing tools can significantly reduce the amount of time spent testing code in comparison to traditional manual methods. Other benefits include the creation of reusable code and a reduction in costs associated with personnel and rework. Get educated on automated testing and available tools to decide if it's right for the QA team.

Conclusions

This article has presented highlights of approaches and solutions for key data warehouse testing challenges. I follow a concept-centered approach expanding on successful methods from multiple respected resources. Testing challenges and proposed solutions described here combine an understanding of the business rules applied to the data with the ability to develop and use QA procedures that check the accuracy of entire data domains – i.e., both source and data warehouse target.

Suggested levels of testing rigor frequently require additional effort and skilled resources. However, employing these methods, DW/BI teams will have assurance from day-one in their data warehouse implementation and data quality. This will build confidence in the end-user community and will ultimately lead to more effective DW/BI implementations.

Testing the data warehouse and BI application requires not just good testing skills but also an active participation in requirement gathering and design phases. In addition, an in-depth knowledge of BI/DW concepts and technology is a must so that one may comprehend well, the end user requirements and therefore contribute toward reliable, efficient and scalable design. We have presented in this work the

numerous flavors of testing involved while assuring quality of BI/DW applications. The emphasis lies on the early adoption of a standardized testing approach with the customization required for your specific projects to ensure a high quality product with minimal rework.

Best practices for the test methodologies presented here have been compiled based on the knowledge gathered from practical experiences while testing BI/DW applications.

Wayne Yaddow is a senior data warehouse, ETL, and BI report QA analyst working as a consultant in the NYC financial industry. He has spent 20 years helping organizations implement data quality and data integration strategies.

Wayne can be reached at wyaddow@gmail.com

Back To Index

Getting Started With UniTEE – Unified Test Engine Exemplified

by Rahul Verma

Introduction

UniTEE stands for Unified Test Engine Exemplified, and is pronounced as **/ˈ juː nɪ ti/** (unity). It is spelled as **Unitee** elsewhere.

It is a test automation engine developed by **Rahul Verma**, with one key goal: **Unification**. It targets unification of different types of test case writing, for different types of test automation, at different layers, in different languages, for different purposes, applied in different development models, for different kinds of products and so on.

Unitee derives some elements of design for its test components from the excellent xUnit architecture by Kent Beck. Some key inspirations are the concept of fixtures, the default assertions and the XML result format. This indicates the impact xUnit architecture has had on Rahul as well as one of the goals of **Unitee** as being consistent in API to what developers & testers are used to for basic operations.

The similarities end here. A fundamental deviation, which Unitee takes, is that **unlike most of the Java based automation frameworks out there, it does not use JUnit as its base. It has been coded from scratch.** It targets keeping easy things as easy as possible and making complex stuff much easier, as the latter is where many frameworks get it wrong.

In short, **Unitee** is a research project to serve **software testers, developers, educationists and students**. The goal is to develop an engine which can provide **an alternative to existing test engines like JUnit, TestNG etc.** This is especially true for the problem contexts, where it aims to serve better than existing engines.

Creating and Running Your First Test Case

Unitee supports multiple styles of writing automated tests. Test Case is the most basic one out of them. Test Case is used to talk about a class that represents one and only one test case. It does not have any sub-test.

The standard way of writing a new test or test suite in Unitee is to inherit from a base class provided by Unitee for that corresponding test style, and override one or more of the inherited methods. As this is a first tutorial for Test Case, we'll keep it simple and focus on writing a Test Case in which we override only one method.

Let's get started and create our first test. The following text assumes that you have configured your IDE (check Unitee reference for **Eclipse**, **IntelliJ** or **NetBeans**). If not, do it before going ahead with this article.

Steps

- Create a Java class with a name of choice
- Your class should inherit from in.unitee.lib.test.TestCase
- Create a run() method in your class.
- Add code that represents your test inside the run() method.
- For this example, we'll use a basic equality assertion.

Following is the example code for a simple Test Case. Replace the package name with the package of your test class. You can find this code in GitHub Unitee-Examples repository **here**.

```
1 package in.unitee.ex.basic;
2
3 import in.unitee.lib.test.TestCase;
4
5 public class SimpleTestCase extends TestCase{
6         public void run(){
7                 checker.assertEquals(1,1);
8         }
9 }
```

That's all. Run it. Play with the code a bit and understand further.

Creating and Running Your First Test Method Suite

Test Method Suite is one of the most common test styles supported by test frameworks across all languages. The style of test writing which we are going to see here, is the default test style writing for xUnit family of test frameworks and the frameworks that use the same as their underlying engine.

Note: xUnit frameworks call this a test case, but Unitee calls it a Test Method Suite. Fundamentally, a test suite in Unitee is a composite test which contains sub-tests. Test Method Suite is a class in which sub-tests are represented by test methods (with a pre-fix of 'test') and essentially the class acts as the container/parent for these tests.

One key thing to remember is that the order of the execution of sub-tests (test methods) is not guaranteed (like xUnit). This enables making your tests independent of each other and can still share the code to take care of dependencies by utilizing Fixtures effectively.

Barring this difference, a Test Method Suite, would look very familiar to you if you have already looked at the Unitee Test Case.

The following text assumes that you have configured your IDE (check Unitee reference for Eclipse, IntelliJ or NetBeans). If not, do it before going ahead with this article.

Steps

- Create a Java class with a name of choice
- Your class should inherit from in.unitee.lib.test.TestMethodSuite
- Create test method(s) in your class. The name(s) of the method(s) should start with the word 'test'.
- Add code that represents your test inside the test method(s).
- For this example, we'll use a basic equality assertion.

Following is the example code for a simple Test Method Suite with three test methods, with each method giving a different result – Pass/Fail/Error. Replace the package name with the package of your test class. You can find this code in GitHub Unitee-Examples repository **here.**

```
1  package in.unitee.ex.basic;
2
3  import in.unitee.lib.test.TestMethodSuite;
4
5  public class SimpleTestMethodSuite extends TestMethodSuite{
6
7          public void testMethodPass(){
```

```
8
9                   checker.assertEquals(1,1);
10          }
11
12          public void testMethodFail(){
13
14                  checker.assertEquals(1,2);
15          }
16
17          public void testMethodError() throws Exception{
18                  checker.error();
19          }
20
21 }
```

That's all. Run it. Play with the code a bit and understand further.

Hope you like exploring Unitee. In the subsequent articles, we would move on to more advanced formats, which are still made simpler in Unitee by its unique design.

Rahul Verma is a consulting software tester, author, speaker, coach and a serial entrepreneur from Bangalore, India. He is the founder of Test Mile and Talent Reboot.

He is known for his practical and unified view of the software testing subject. He has been awarded multiple Testing Thought Leadership awards for his contributions to software testing community.

You can visit his website www.rahulverma.xyz to know more about his work and get in touch.

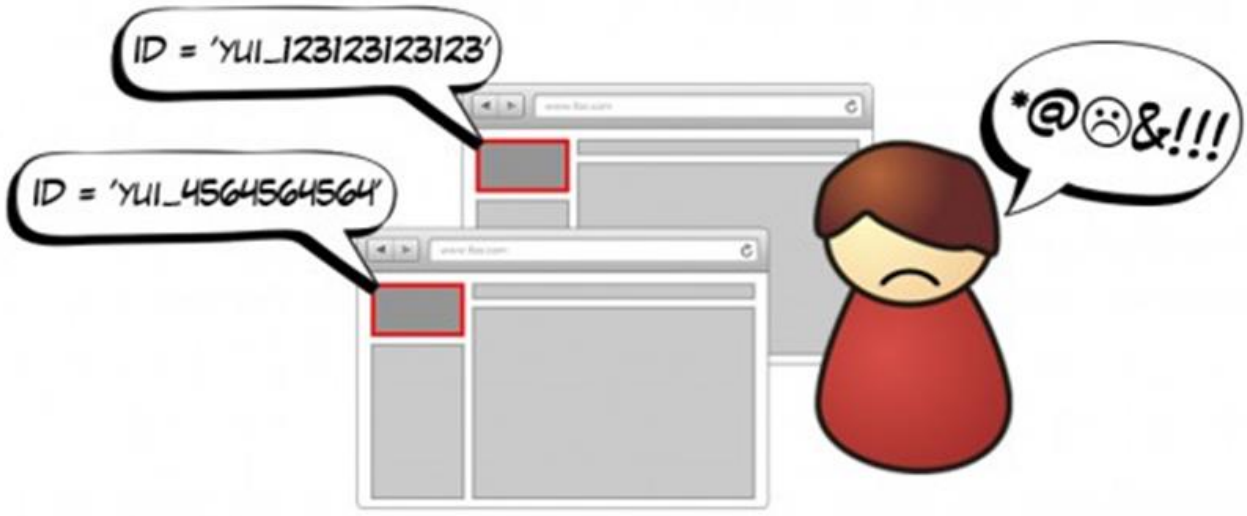
Back To Index


From Special Desk

Automated Testing and Dynamic IDs

- by Tobias Walter

If software is configurable, it often means that the software relies on dynamic content – dynamic content is typically based on dynamic identifiers (IDs). Using dynamic IDs often leads to problems in test automation because they are newly generated each and every time an element is displayed. This blog post will illustrate how easy it is with Ranorex to overcome this frequent problem in test automation in an automated way.

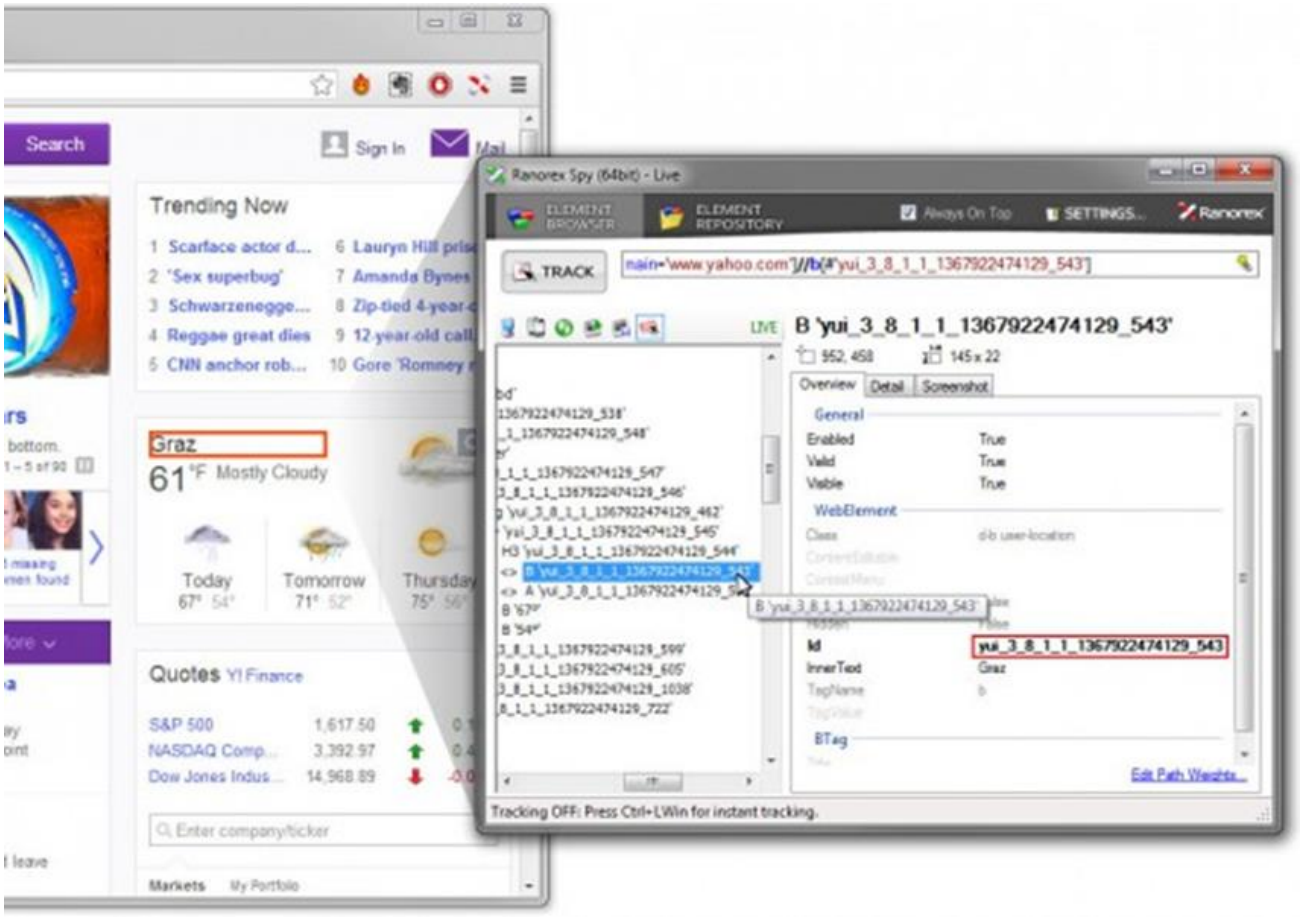


What is the Problem

Ranorex decides which attribute will be used for object identification based on predefined attribute weights (or "RanoreXPath weights"). Usually the ID of an object will be the best and most stable attribute for identifying and object and therefore has the highest weight. But when facing dynamic IDs, the ID cannot be used to identify an element anymore because it changes each time the element is reloaded/displayed. That means that you manually have to change how the objects are identified in your repository (which might be previously created by the recorder). This blog post will illustrate how you can add so called "weight rules" (RanoreXPath weight rules) allowing you to continue using the recorder and no longer changing your repository manually afterwards.

Identify Dynamic Content

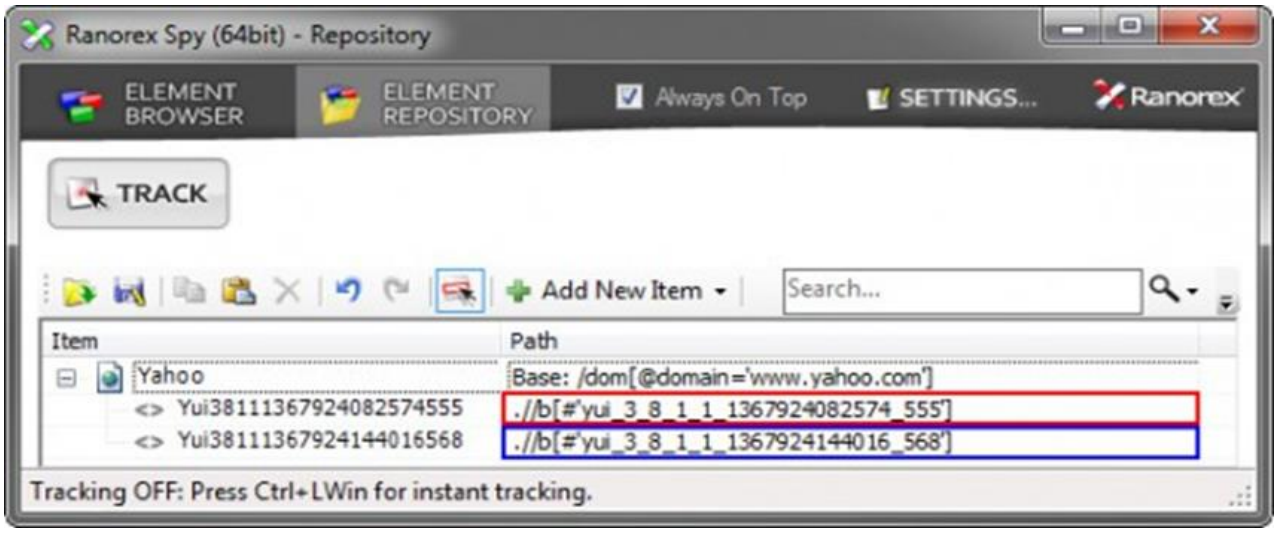
First of all we have to identify in general whether we do have a dynamic ID problem or not. Let's have a look at yahoo.com which is based on YUI (Yahoo User Interface library), a free open source JavaScript and CSS library. This library uses dynamic IDs by default. To analyze the occurrence of dynamic contents you can use Ranorex Spy from the start menu. Use the "Track" button to analyze an element of the web page.



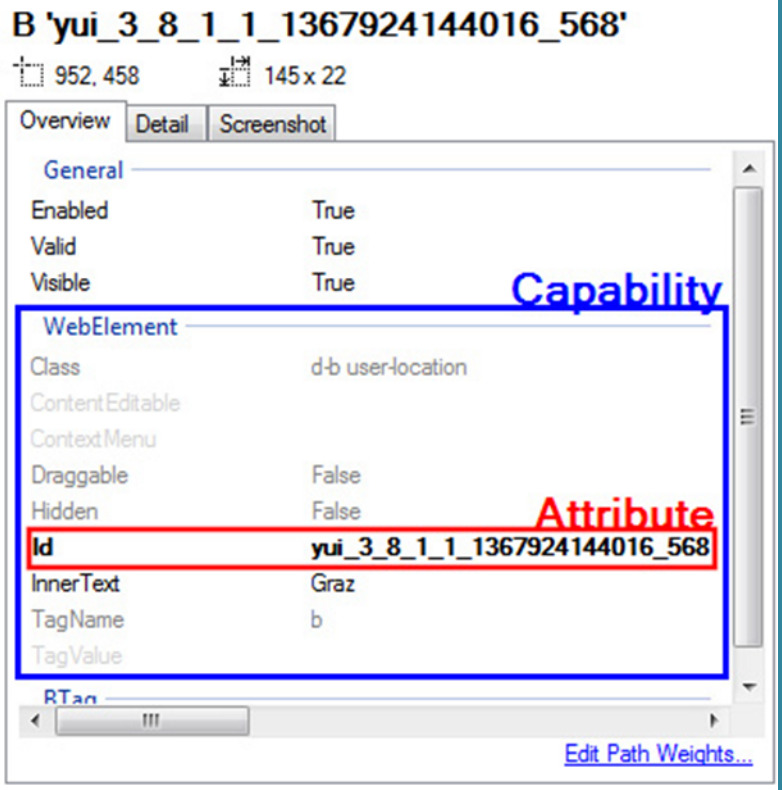
When you take a closer look at the attribute "Id" of the just tracked element, you will see that it contains dynamic content (yui_3_8_1_1_13679224741219_543).

That leads on the one hand to a poor readability, and on the other hand destroys the testability of the page as the element cannot be identified by its ID anymore after reloading.

To illustrate this behavior, add the analyzed element to the repository (using the context menu), reload the website and add the element again. You will see that the same element will occur twice in your repository having different RanoreXPaths based on different IDs. Highlighting the first (red) repository element will not work as an element with the given ID is not available anymore. Also, the second (blue) element will only work until you reload the page again and a new ID is generated.



So this way we identified that we do have a dynamic ID problem. (For the sake of completeness: In most of the cases the attribute "Id" leads to dynamic content issues, but of course there can be other attributes creating identification problems as well.)

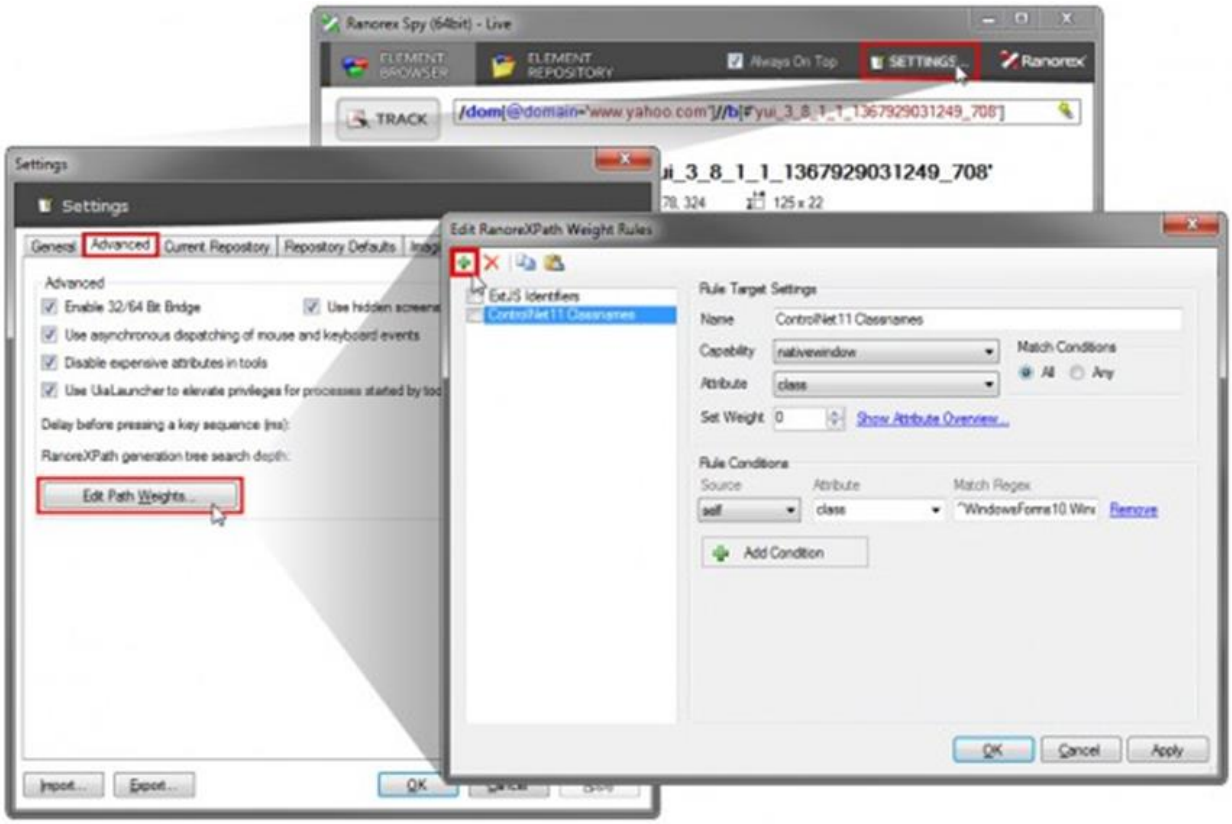


Add a Weight Rule

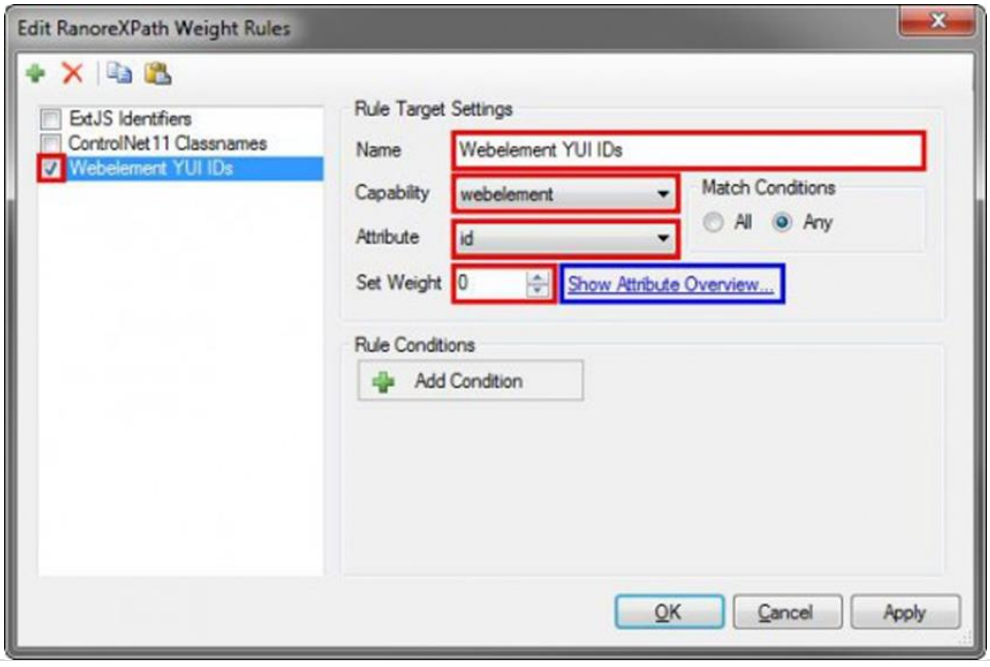
What we know now is that we will run into a problem when choosing the attribute "Id" to identify elements. Now we have to identify where our dynamic ID belongs to and create a rule on the base of that. As we can see in the "Overview" tab of Ranorex Spy, it belongs to the capability "WebElement":

Write down the capability and the belonging attribute.

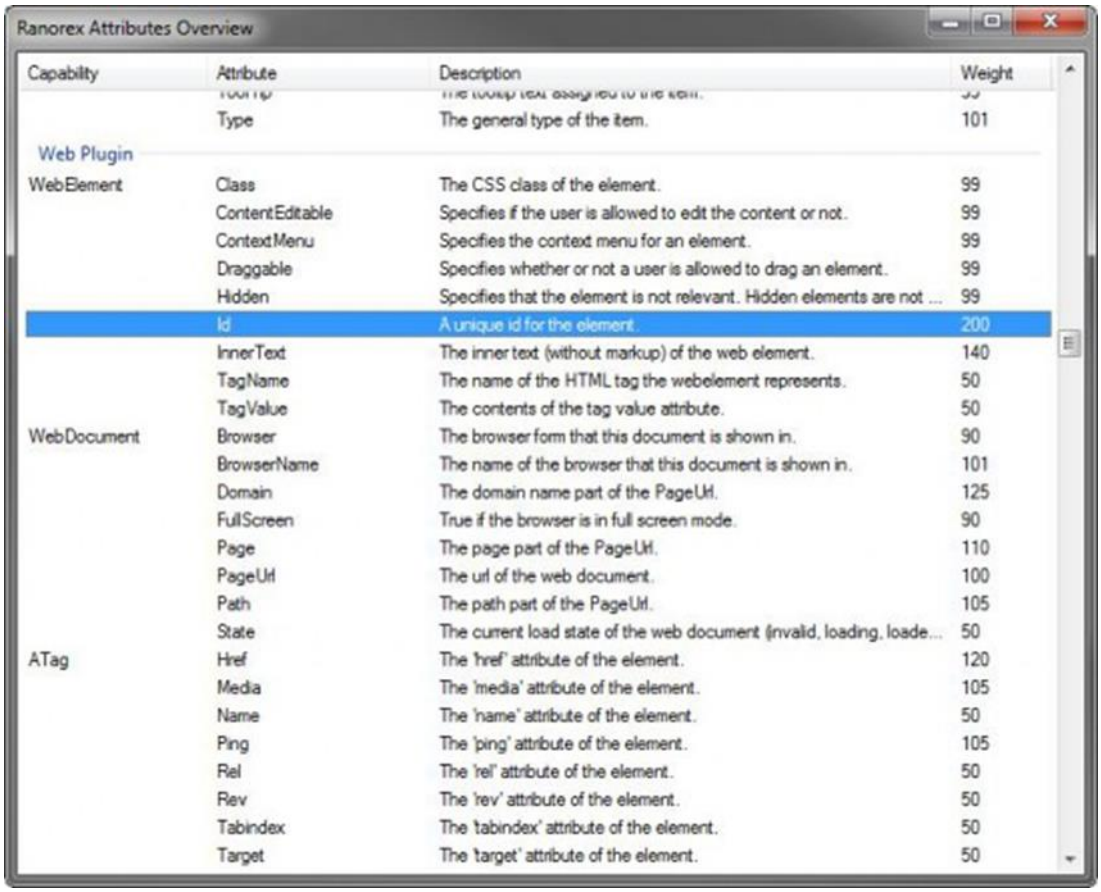
After identifying the correct capability and its attribute, we add a path weight rule to "filter out" the dynamic content and use another property for object identification. To do so, make sure that all other Ranorex instances (Studio, Recorder, Spy, etc.) are closed and only the currently using Spy instance is open. Then open "RanoreXPath Weight Rules" editor from settings dialog and add a new weight rule by pressing the "+" button:



Choose an appropriate rule name, select the previously identified capability as well as the attribute, and set the weight to 0:



Have a look at the attribute overview ("Show Attribute Overview…") to get a feeling for the existing attribute weights. In order to navigate to a specific capability, simply press the initial letter of the capability's name on your keyboard.



As you can see in this dialog, the weight of the attribute "Id" is set to a value of 200. When lowering this weight to "0" the attribute "InnerText" will be used to identify the element as it's the next higher value of 140. If no other attribute is available (because the weight is set to zero or it has no value) the index of the element will be taken to identify it (e.g. [1]).

(For the sake of completeness: In addition to lowering the weight of attributes you don't want to use, you can also raise an attribute's weight to favor the use of this specific attribute for object identification.)

Add a Condition to the Weight Rule

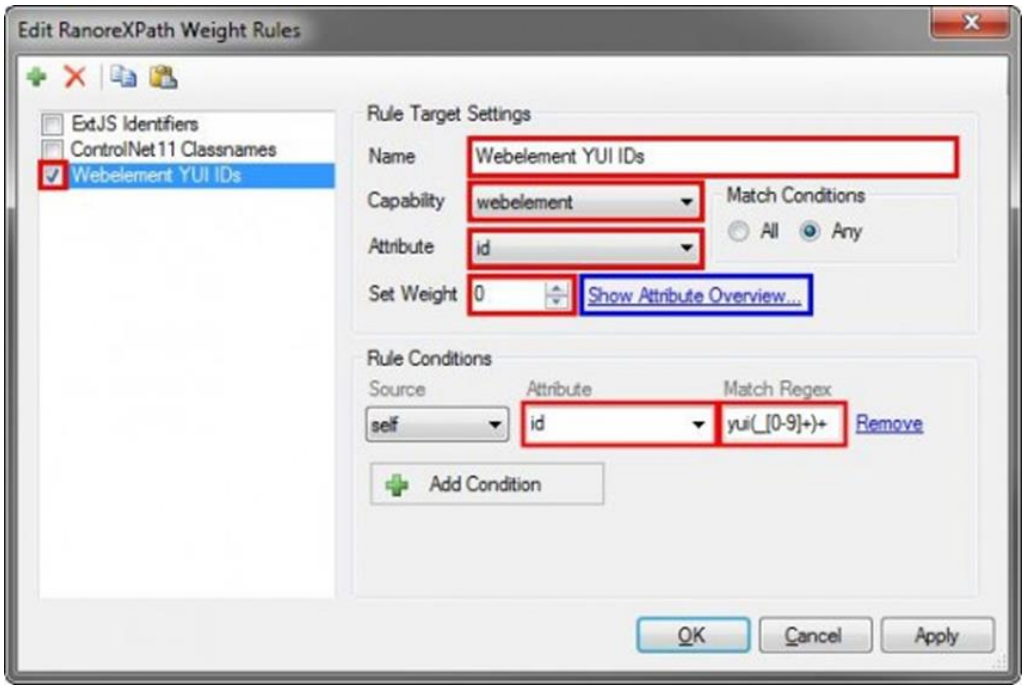
By lowering the weight of the attribute "Id" for the capability "WebElement", the attribute "Id" will no longer be used for object identification in general. This rule will not only filter out dynamic IDs on yahoo.com but also all other IDs for all other web pages which might not be what you want to achieve.

To overcome this behavior, you can add a condition by pressing the "Add Condition" button. This defines which IDs should be filtered and which shouldn't.

You might have found out already that dynamically generated YUI IDs always have the prefix "yui" followed by "_" and a number for x times. This can simply be represented by the regular expression

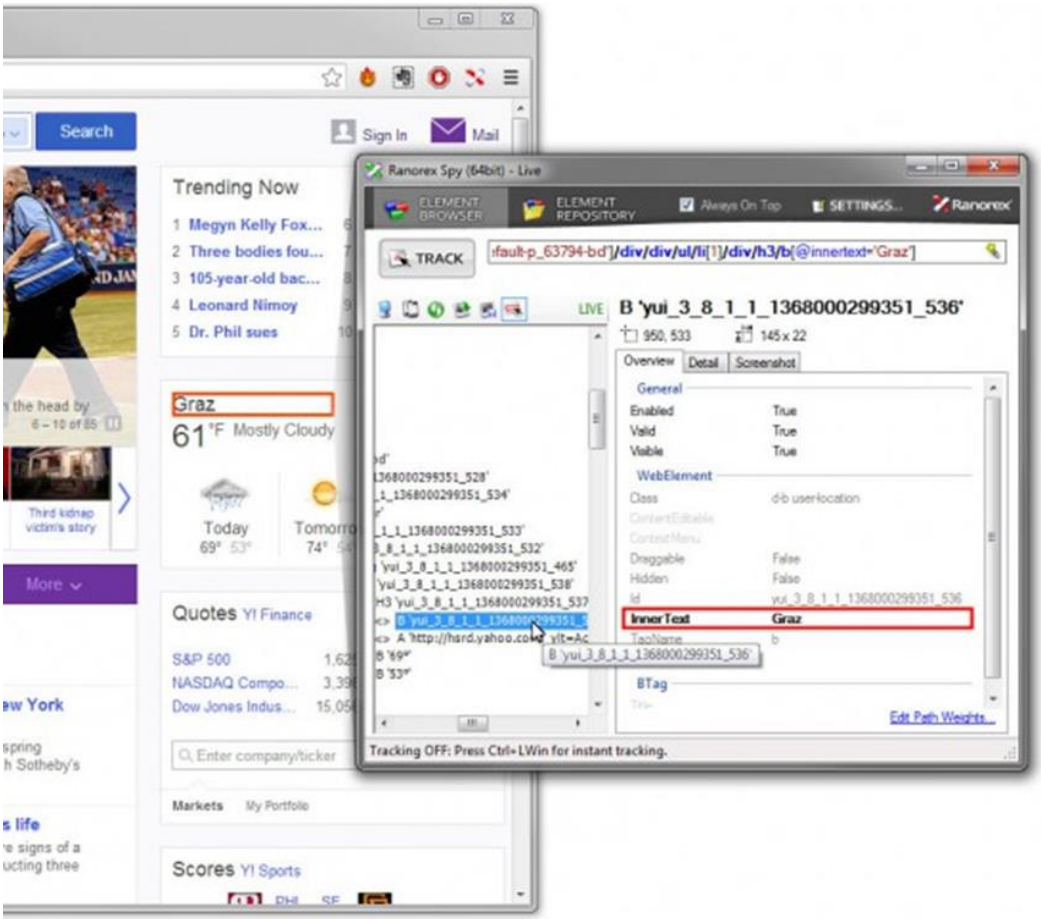
yui(_[0-9]+)+

The matching expression must start with the string "yui" followed by "_" and a number for one or more times. The whole "_" and number stuff must occur one or more times. For further details about regular expression have a look at the regular expressions wiki page or the chapterRanoreXPath in our user guide. Select the attribute you want to match against the regular expression from the dropdown, and then enter the regular expression.

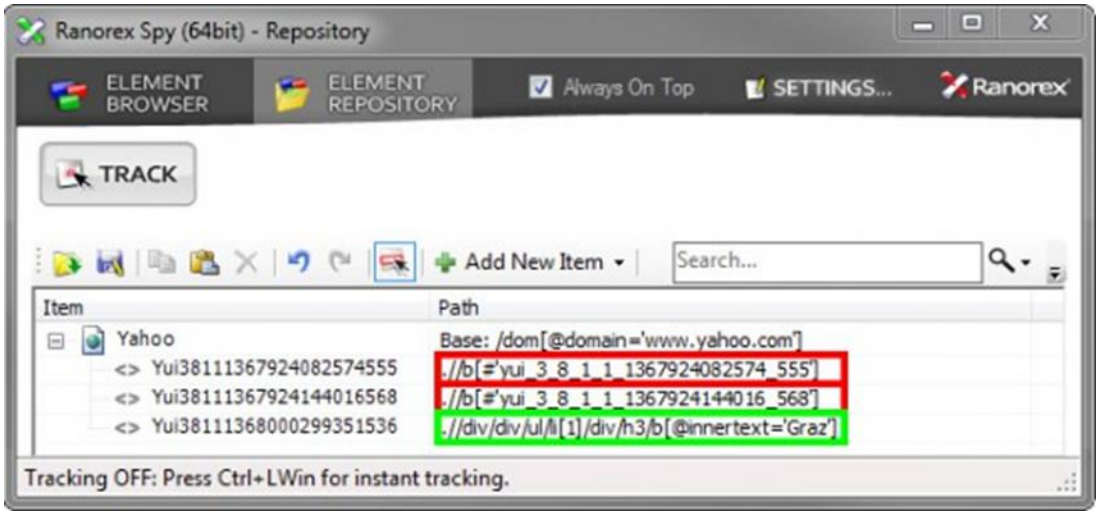


Check whether the newly created rule in the left sided list is enabled, and save your changes by pressing the "OK" button.

You can check whether the newly added RanoreXPath weight rule works or not by tracking the element again on the website. You will see that the attribute "Id" is no longer used to identify the element, but by the "InnerText" attribute now.

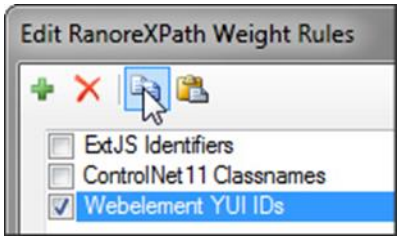


You can best see the difference by adding the element to the repository again (green).



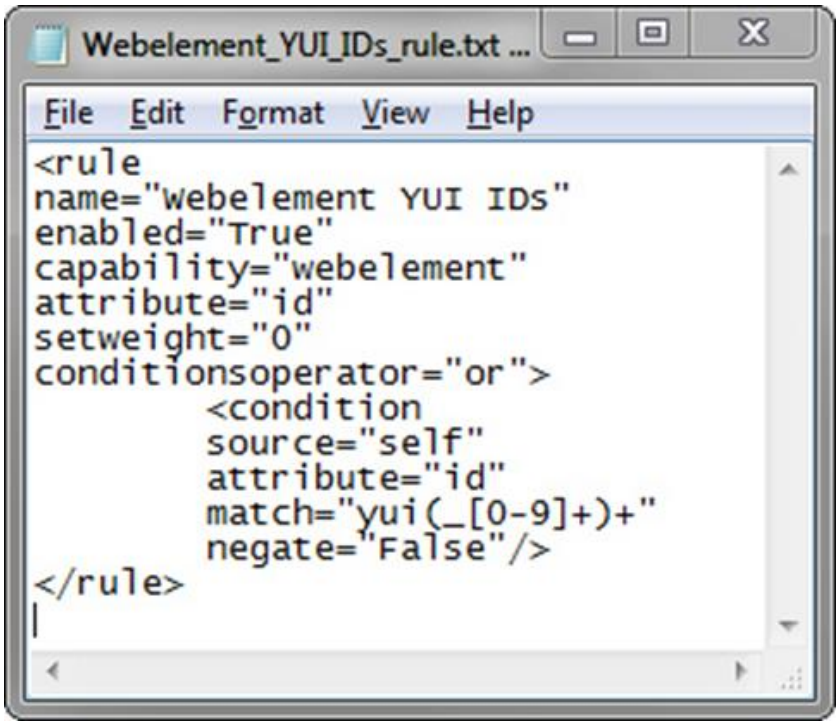
Share the new Rule with your Team

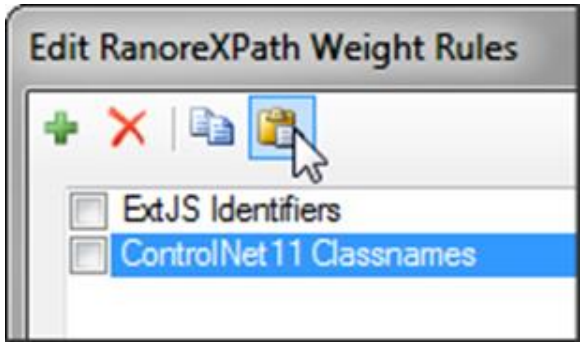
Once you successfully created a new rule, other colleagues testing the Yahoo web page might be interested in it, too. You can easily share the rule by using straightforward Copy and Paste functionality. Therefore copy the rule to clipboard via CTRL + C or by using the "Copy" icon in the toolbar.



Then paste it to the text editor of your choice (e.g. Notepad) by pressing CTRL + V and save the file with a meaningful name, e.g. Webelement_YUI_IDs_rule.txt.

On the target machine of your colleague, copy the entire file content (CTRL + A, CTRL + C) and paste it to "RanoreXPath Weight Rules" Dialog by pressing CTRL + V or use the toolbar icon "Paste".





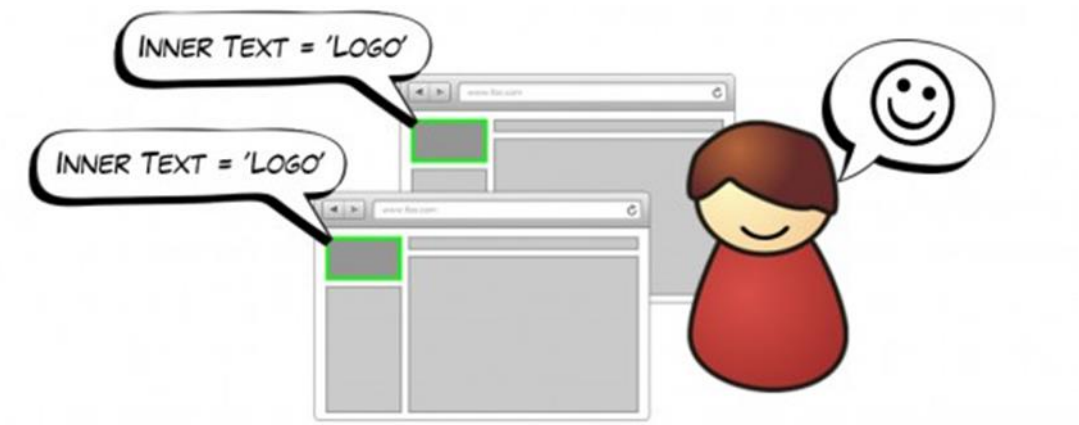
Conclusion

Using the RanoreXPath Weight Rules can assist you in automatically creating a robust repository which is the fundament of a robust test automation framework.

Also, it gives you the great advantage to continue using the recorder, and no longer being forced to change all your dynamic UI-elements in the repository manually. Just re-record and you will be fine.

By sharing the rule with your team, other testers save time and can also directly start recording.

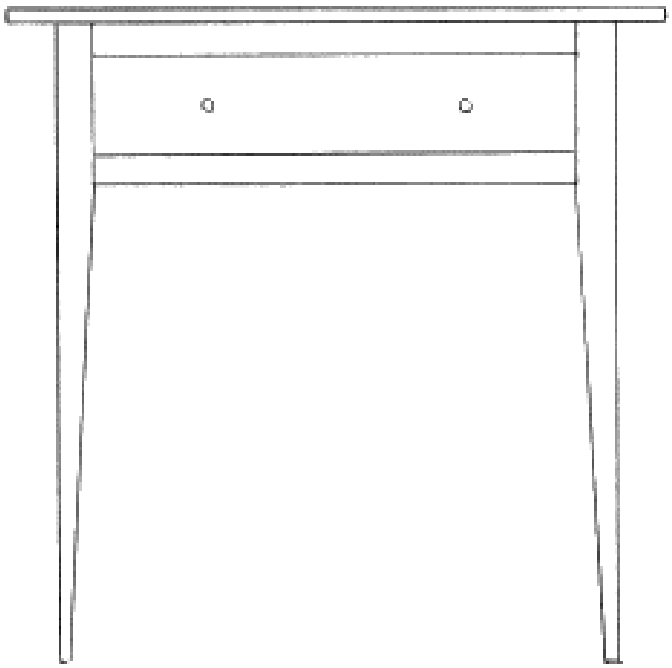
Since you created a global "path weight rule" for your object identification, the rule will also be applied if you create a repository manually (as described in "Did you know… that you can manually generate a recording?").



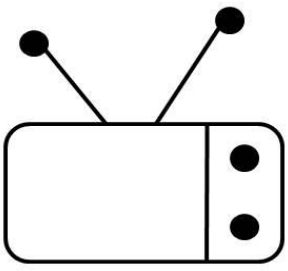
Author: Tobias Walter

Tobias Walter is Online Marketing and Content Specialist at Ranorex GmbH. Next to his marketing activities he also writes articles published in several blogs and journals.

Back To Index



TV for Testers



Your one stop shop for all software testing videos

Sharing is caring! Don't be selfish ☺

Share this issue with your friends and colleagues!

CONTINUED FROM PAGE 11...

Is this unique to testing?

I hear this question most often from freshers, people who have recently gotten a job in testing or who are trying to get one. My impression of the typical person who asks this question is of someone who is intelligent, ambitious, willing to work hard, maybe university educated but without many skills that are well enough developed to be of practical value in the workplace. This person is at the start of a journey of self-improvement that will require many years of work.

Every field repeats it's past, cycling through the same few basic strategies for addressing its basic problems.

This is not necessarily a bad thing. If you have new knowledge, if you have solved a few other problems, then an approach that didn't work the last time you tried it might finally work.

However, all areas of computing seem to suffer from amnesia. If an idea hasn't been popular in the last ten years, very few people will remember how popular it was before that. Honest people can honestly sell the idea as "new" because they are so ignorant of their field's history. We spend very little time reading the history of what we do.

Some consultants get very impatient with me when I tell them that they are selling something as "new" that is very similar to an old idea. Because they didn't learn from history, they had to reinvent the idea, or learn it from other people who reinvented it. They don't see the similarities. And the superficial things are different. The words people use to describe the idea are different. The words people use to describe the problem that this idea tries to solve are different. The social context—how programmers and project managers and marketing people and customers work together—is different too. For someone who doesn't want to recognize a historical pattern, these superficial differences make repetition easy to ignore.

This kind of problem is more general than testing, more general than computing. I see it running through the social sciences, through psychology and economics for example. However, I think we are less well read in computing than in the social sciences, less familiar with our history.

Why is this a problem?

If you don't realize that you are trying an old approach, you won't ask why that approach failed the last times people tried it. You won't revise your version of the approach to avoid the old mistakes and to overcome the old barriers.

You started by talking about how testers progress more slowly than programmers, but here you are saying that testers and programmers have the same problem. What is the difference?

I think that programmers and testers are similar in that both groups are blind to their history. This isn't just a problem among practitioners. University degree programs, at least the typical programs in North America, are much more focused on the current ideas, the current technology, the current practices, than on the history of them or the controversies that led to the evolution of the current forms.

What is different about programmers is that they incorporate what they are learning into their technology. They create new tools, new libraries, and new languages. They package what they are learning into a form that other people can use and that will still be usable (and useful) even after the ideas that inspired the technology have (again) become unfashionable.

The other difference, I think the essential difference, between the evolutions of programming and testing is that programmers don't just incorporate what they learn into their technology. They incorporate their technology into what they learn.

What do you mean?

When we teach people how to develop software, we don't just teach them "about" programming. We teach them how to actually do programming. We give them programming assignments. The assignments might be fairly easy in the introductory programming course but they get harder. To do the assignments, the student must learn more skills, work with more tools, and learn new ways to solve problems. We intentionally give students assignments that force them to try new approaches, to learn new ways of thinking about computing.

Students don't just study "programming". They take courses like "data structures", "algorithms", "design methods", "user interface programming", "user interface design", "computer organization, machine language and assembly language", "thread-safe programming", "network design", "design of programming languages", and so on. They learn many languages that demand that they think about problems in fundamentally different ways. The typical university computer science student learns an object-oriented language (like Java), a traditional structured language (like C or FORTRAN), an assembler, a scripted programming language (like Ruby or Perl), and perhaps a functional language (like Haskell). They learn to write code, and use development tools, in at least two operating systems. They learn to rely on programmers' libraries, to find new libraries online (essentially, new collections of commands for their current language) and to save their own good code into libraries that they can reuse on later projects.

We don't go into this type of depth in testing.

Is that because we don't have university degree programs in software testing?

No.

Then what's the difference?

Most commercial courses in software testing stick to easy topics. They teach definitions. They teach corporate politics and let people discuss (complain about) their situation. They present simplified examples. If they give students in-class assignments in class, the assignments are designed to be straightforward, so that everyone can finish in a reasonable time. Very few classes have homework and of those, very few have homework problems that are hard. Testing courses make you feel good.

We have a remarkable number of introductory courses in software testing. People drift from one course to another, thinking they are learning exciting new things when they're really getting a new batch of vocabulary, exposure to a new collection of political attitudes (attitudes about allocation of power and responsibility in software projects), a superficial introduction to ideas from some other field (maybe psychology, maybe engineering quality control, maybe statistics, maybe programming, maybe something else), and a collection of fun stories and fun activities.

But people come out of these courses believing that they've learned a lot

It is easy to foster the illusion that we are covering important topics.

For example, we can have arguments that feel deep and meaningful about the proper definitions of words. We can introduce "profound" new definitions for familiar words and "deprecate" the old definitions. I don't think definitions are important. I think we have conflicting definitions in the field and so we have to learn to listen to each other. I think that debates about the "right" definition lead nowhere. However, they can give the participants the illusion of progress. People can feel as though they've done something important even though they haven't done any real work and they haven't learned how to actually do anything differently.

As another example, we can work through simple examples. For every testing technique, there are simple, straightforward examples. To teach this way, you describe the technique in lecture, then work through one or two examples in the lecture. Next, pick an example that is very similar to the lecture. You can make it sound different by changing the setting—for example, you can design a test for the boundary of the size of a data file instead of the size of an input data item, but both boundaries can be integers that are specified as part of the problem. The students will have a success experience and feel confident that they know the technique. They'll feel ready to move on to the next technique because they won't realize that they probably can't apply this technique to problems that are even slightly more difficult.

As a third example, we can show how something in our field is like a problem studied in a different field, and then look at a model that people developed in that other field. I think this is a good start on teaching people to expand their thinking, but what does the course do with it? Do the students ever actually use the model? Do they learn to use it on the job? How much practice do they get in applying it?

If the course doesn't give students enough experience with a concept or model, if it doesn't give them experience actually using it under circumstances that are similar enough to the student's work-life experiences, then the student won't be able to apply it when they get back to work.

We have plenty of experience going to places where a person at the front of the room gives you information that you tell people about but you can't make practical use of it. We have a special name for this type of information. We call it "entertainment."

There's nothing wrong with going somewhere for entertainment. But don't confuse it with education or training.

If you're going to spend your time on topics that are presented so superficially that you won't be able to apply what you've learned, from a what-are-you-learning point of view, you'd be as well off (and probably better entertained) listening to jokes and watching magic tricks.

Are you saying that people should avoid introductory courses?

You have to start somewhere. It's good to have an introduction.

But an introduction is just an introduction. It's a starting point. It's just one small first step on a path to competence in a field.

But if they take the right course, people can write an exam and be certified after an introductory course.

If you can learn what you need for a "certification" from an introductory course, that certification is not a measure of your knowledge. It's a sales tool for the people who are selling the introductory course.

The fact that you can get "certified" after a few days of training is one of the problems in our field. This tells people that you can learn the basic knowledge and skills of the field in three to five days. That you can take this course, and then pass this multiple-choice exam, and then you can stop for a while because you have the basics.

This illustrates the difference that I was talking about between testing education and programming education. If I told you that I could teach you how to program in three days, well enough to get a job as a programmer and keep that job, you would probably laugh at me. Anyone who is any good at programming knows that you can't get the basic skills that you need in a matter of three days. And that you can't demonstrate your skill as a programmer by answering simple questions on a multiple-choice exam.

We should know the same thing about testing, but as a field, we don't. A huge portion of our field takes training that is specifically designed to prepare people for these silly exams.

But some certification exams are hard to pass. Doesn't that mean they have high standards?

No.

Why not?

The pass rate tells you very little about what the students actually learned, how useful the material was, or even how hard the material was.

When I teach a course at university, I can make my exams as easy or as hard as I choose. If I want to pass 10% of my students, I can write an exam to do that. If I want to pass 90% of my students, I can write an exam to do that too.

I can fail 90% of my class even if the underlying material is easy and I can pass 90% even if the course material is very hard.

Learning how to set the difficulty of an exam is one of the skills that professional teachers learn.

Do you think the people who teach the primary certification course are dishonest?

No.

I think they (ISTQB, ASQ, and QAI) have the wrong vision of testing education but I think that they are trying very hard to provide a good service (good courses, fairly-written exams) that is consistent with their vision.

I know much of QAI's history but I am not familiar with the quality control processes of the new owners of QAI, so I am not comfortable making additional statements about QAI.

Regarding ISTQB and ASQ, I am very impressed with the efforts they make to evaluate and improve the quality of their exams. I respect the professionalism of their work. I just think they're taking the field in the wrong direction.

How do you think testing education has to change?

That's what I came to university to think about. I was unhappy with the impact of my teaching as a commercial instructor. I felt that my colleagues and I were providing too much entertainment and too little skill development, that we were making too small a difference in the lives of our students.

So I joined the computer science faculty at a good school, to see how they taught novice programmers to win programming competitions and land good jobs. As I learned the basics of university teaching, I proposed a project to the National Science Foundation, to combine what I had learned from commercial training in software testing with good practices in university-level teaching of mathematics and programming. The NSF gave me three grants, over 10 years, to develop my approach. This is the work that led to the BBST course series (http://bbst.info).

Would you like to tell our readers about your overall experience with BBST courses?

Rebecca Fiedler and I designed these courses, with a lot of help from other people. Becky had been a professional teacher for almost 20 years—while she worked on BBST, she also did other research that led to her Ph.D. in education and she became a professor of education. Much of the BBST instructional theory (as opposed to the testing theory and practice) came from Becky.

Our goal is to use technology to draw students into a deeper and more intense learning experience, helping them understand complex issues, develop complex cognitive skills, and develop real-world skill in the use of key software testing techniques.

One of the weaknesses of commercial training is that it rarely involves significant assessment. "Assessment" means measurement of what the student knows or can do. To assess student work, we have do design good tests, good projects and so on.

Good assessment serves four purposes:

a) Assessment tells the students what they do and don't know
b) Students learn from participating in the assessment activities. People learn what they do. The assessment activities structure what they do, and thus what they learn.
c) Assessment tells the instructor what each student knows (or doesn't know)
d) Assessment tells the instructor what parts of the course are weak.

When you're teaching a course, assessments force you to confront the weaknesses of your work. There are many excuses for not doing assessments or for making assessments so easy or so simplistic that you don't learn much from them. However, if you want to improve as a teacher, studying what your students have learned is how you discover your effectiveness.

Our typical student spends 12-15 hours per week on each BBST course for four weeks (48-60 hours). Of that, about 6 hours is lecture. The rest is the student's work: doing assignments, participating in discussions, preparing for and writing quizzes and exams, and processing the feedback they get on their work. People learn what they do. We provide a structure in which students do a lot and (those who do) learn a lot.

What are the students learning?

In the first course, BBST-Foundations, students learn the basics of black box software testing. This is our version of the introductory course. We see it as the starting point for the BBST educational experience, not as the primary experience. The next course, BBST-Bug Advocacy, is my favorite course. Students learn how to report bugs effectively. They learn a lot about

- troubleshooting (how to demonstrate the failure and its consequences)
- market research (how to demonstrate that some aspect of the program's design reduces the program's value)
- persuasive technical writing (how to describe the bug precisely and in a way that motivates people to fix it)
- decision theory (how people make decisions about the importance of the bug and the value to the project of the bug reporter (i.e., you))

The third course, BBST-Test Design, is a fast march through the field's main test techniques. We peek at about 100 different techniques and focus on six. We do small projects using two of the techniques and larger, harder projects on two others. The course overwhelms many students with work—we provide hundreds of references, cross-referenced to the course topics, so that students can come back to individual topics later, when they need to apply one on the job.

Bug Advocacy and Test Design convey knowledge that is more practical—how to do the work of testing—while Foundations presents an overview.

We would like to know about your book on Domain Testing. Why did you decide to write a book dedicated to single test technique?

The Test Design course introduces students to several techniques and gives them a brief experience with them. We try to give experiences that are realistically complex for a few techniques, including Domain Testing, but one course that surveys many techniques, can only go into limited depth.

The Domain Testing Workbook goes into depth. This is not just an introduction to our field's most commonly practiced technique. Our goal is to help readers develop professional-level skill. The book's 450 pages present the overview and theory of the technique and then work through the analytical process in detail. What kinds of information do you use, and how do you use it, to create an excellent set of domain tests? We write the book around 30 worked examples that present different issues and challenges that someone who regularly uses this technique will have to face. For more on this book, see http://www.amazon.com/Domain-Testing-Workbook-Cem-Kaner/dp/0989811905.

Will this turn into another BBST course?

Yes. We wrote this as a textbook for a BBST course, and created BBST-Domain Testing to present its material.

Your first three courses used videos and slides only, with a few readings but no textbook. Is it different working with a textbook?

The impact of the Domain Testing Workbook on that course was overwhelmingly positive. It inspired us to create workbooks for the other BBST courses. The Foundations book is published now (see http://www.amazon.com/dp/0989811921). The workbook for Bug Advocacy is almost done. We use the draft version in our Bug Advocacy course. The final version of this book and the Test Design Workbook will be published by the end of 2015.

These books significantly update the courses, providing a new set of assignments and a detailed commentary on the lectures. We're now recreating the video courses that reflect the lessons we've learned teaching from these videos over the past ten years.

Do you expect to write new in-depth workbooks like Domain Testing?

I hope so. Becky and I have been writing sections of the Scenario Testing Workbook for years and I've been accumulating notes toward the Risk-Based Testing Workbook and the Specification-Based Testing Workbook.

Why are these in-depth presentations important?

These deeper books illustrate what I think our field most needs. We need more training on:

- how to design great tests,
- how to run them,
- how to evaluate the results,
- how to report the results, and
- how to use computational skills to automate more aspects of these tests and to implement testing methods that are too hard to do manually.

I don't think we need more surveys that talk about these topics. We need to create courses that teach the nuts and bolts of how to do these things, how to do them well, and how to do them efficiently.

This is the foundation that I think we need to make fundamental progress in the state of the practice in software testing. The feuds and the definitional wars are distractions—marketing tools for the businesses that promote them—that I think are increasingly irrelevant to the development of high quality software. We need to improve our skills and our efficiency, not our politics.

So...that was Dr. Cem Kaner on various topics around Software Testing. We hope you liked the series. See you next month with another awesome interview…

Happiness is….

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

Like our FACEBOOK page for more of such happiness

https://www.facebook.com/TtimewidTesters

Slice Carefully or You will be Hurt!

This is about two real situations that I encountered with two different customers. Both these customers were keen on optimising the test cases which they felt was way too much. I discovered that the problem was in the way the system was "sliced" with the test cases being designed for these slices. In the first case, poor decomposition of the system resulting in very long 'vertical slices' resulting in too many cases, while in the second situation, the system under test was sliced into 'short horizontal slices' that resulted in test cases that became unwieldy to handle with time.

In both these cases, the software is not new, it has been in use for a few years and with time the test cases become large and unwieldy and hence the focus on optimisation. Now onto the two situations…

Situation #1

In a recent engagement with a customer, where we had to optimise their test cases, the problem presented itself rather quickly. As we examined the specification of a feature that in addition to the description had a detailed flowchart that clearly prescribed its behaviour.

And my eyes popped looking the humongous flowchart! It had over a hundred conditions. Using this to generate test scenarios by traversing each path would create a mon-ster. Yes it was a complex feature, but this seemed to be way too much. It was time to dig in and analyse.

On analysis, I discovered that this feature used a set of modules to do some work which in turn called the next level set of modules and that the flowchart took into consideration all the conditions of the modules that were called including the ones below it too! In short the behaviour had been flattened to include the lowest level behaviour too. No wonder he gazillion conditions. The system was sliced so deep and all it did was to create a deep gash that hurts!

What was supposed to be tested? Well it was supposed to be a 'Feature'. But it was really the feature and all the lower level components/modules together. No wonder the test cases were too large in number and spaghetti like.

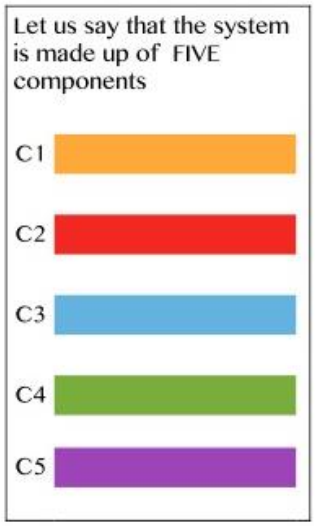
Situation #2

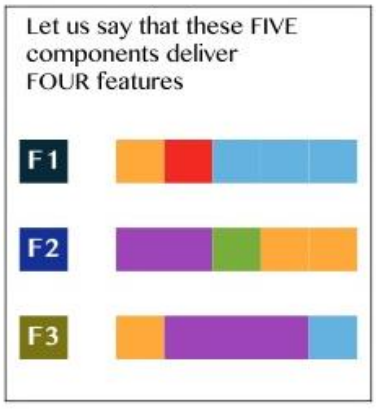
In this case, the large system is being continuously being updated twice a year. The re-leases consists of set of change requests (CRs) that are implemented, tested and re-leased. These change requests are the basis for design of test cases and therefore it is these test cases trace to. The change requests are either due to enhancements/modifications or due to field issues.

So what is the issue here? As CRs came in, new test cases were developed by the customer to validate these changes. A change request could be in a component, feature or a flow (collection of features). Test cases were designed for each CR and with time the number of test cases became non trivial. To qualify the system for a release, new test cases for new CRs were designed and executed and then impact of changes analysed so that appropriate prior test cases could be run to regress the system. And the latter was the problem! Picking up the prior test cases that were traced to previous CRs and not to a basic element like component/feature/flow posed a challenge, requiring a skilled person who remembered the past CR details to (1) identify which CR to look into for the test cases and then (2) pick up those that need to be regressed. It became person dependent, which otherwise demanded one to execute a larger set of prior test cases due to difficulty in selecting a smaller subset.

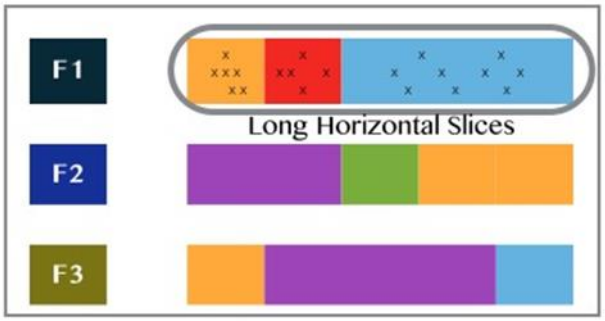
So what is the problem? In this the 'deltas' i.e. changes done to a components were aggregated to form a CR that was traced to a feature or to a flow. The system was sliced into a series of 'deltas', not by components, features, requirements or use cases.

Let me illustrate this via pictures…

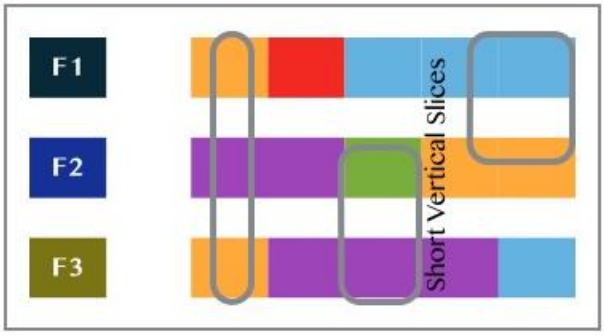




Consider a system that is made up FIVE components that combine to deliver THREE features which in turn combine to form two requirements/use cases. Note that this illustrates a good break decomposition of a system **i.e. sliced properly.**



In Situation #1, using the above picture, the system seems to be sliced into long slices where the behaviour conditions of each component are combined to form a long horizontal slice with the feature's behaviour described as the aggregate of all the conditions. Now the feature is complex and unwieldy as this is no more a feature but a feature plus with the gory individual behaviours of the components combined. And therefore this form of slicing where all the detailed conditions of the component are aggregated makes it complex.



Now let us analyse situation #2. In this what should have been a Feature F1/F2/F3 is missing! What we are see are short vertical slices that represent the changes to components/feature which are combined to form a CR. And test cases are designed to validate the changes represented by the CR. Since we do not see a clear notion of Feature or Requirements/Use case, test cases are traced to CR. And regression is a pain here, as we have understand how the changed components in a CR can affect the other components that may be part of prior CRs. So selection of test cases for regression is an issue in this case.

In Situation #1, optimisation can be achieved by proper horizontal slicing of modules and specifying the behaviour of the whole entity (feature). In the second situation, slicing horizontally is an issue, as the system has not been decomposed into its elemental constituents i.e. components/features/requirements/use-cases.

Setting a clear baseline of "what-to-test" is critical to ensure good optimal design and ensure good maintainability that facilitates a logical choosing of the minimal tests for regression. It is about how you slice the system. Do it wrong and you have too many cases the first time to execute or too many test cases to regress! It hurts in both these cases.

Slice carefully!

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

International Conference on QA & TEST Embedded Software Testing

14th International Conference on Software QA and Testing on Embedded Systems

This year we offer a special program: **3 Keynotes, 2 Tutorials, 8 Tracks and a Special Lecture**, from renowned companies such as: **Intel, Samsung, GMV, Robert Bosch, Dell** and **Verifysoft.**

QA&TEST, the international Conference on Software QA and Testing on Embedded Systems, will be held on 14, 15 and 16 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

Special offer 20% discount using the code TEA20

Organiser

SQS SOFTWARE QUALITY SYSTEMS

Sponsor

innovalia ASSOCIATION

October 14 · 15 · 16

2015

Bilbao · Spain

TESTBASH GOES TO AMERICA!

TESTBASH NYC

ON NOVEMBER 5-6TH 2015 WE WILL GATHER TO HEAR, LEARN FROM, MEET AND SPEAK TO OTHER PEOPLE WHO LOVE SOFTWARE TESTING.

TESTBASH IS A CONFERENCE DEDICATED TO THE ART OF SOFTWARE TESTING. WE ARE AN OPEN AND FRIENDLY COMMUNITY OF PEOPLE WHO ARE DEDICATED TO IMPROVING THE WORLD OF SOFTWARE TESTING.

WE ARE A FRIENDLY BUNCH. COME JOIN US. YOU'LL SOON FEEL AT HOME.

"I LOVE TESTBASH BECAUSE OF THE EVENTS AROUND IT ARE OUTSTANDING, #99SECONDTALKS CHANGED MY LIFE, AND THE ATTENDEES (I'VE MADE ACTUAL FRIENDS THERE!)" - VERNON RICHARDS

WWW.MINISTRYOFTESTING.COM

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.



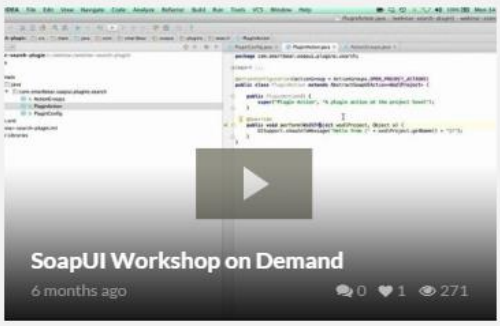
We enjoy putting our experience to your service. Our know-how allows us to provide consultation services for projects at any stage, from small up to super-large. Due to our range of expertise we can assist in software architecture and COTS selection; review of software designs; creation of testing suites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

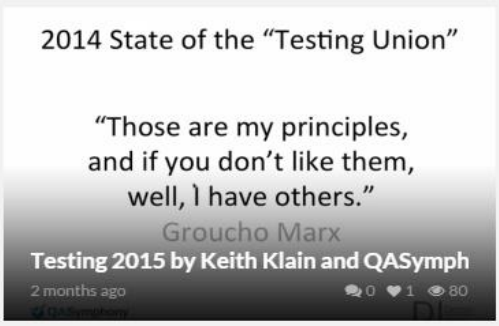
look us up: www.testomaton.com - twitter: @testomaton

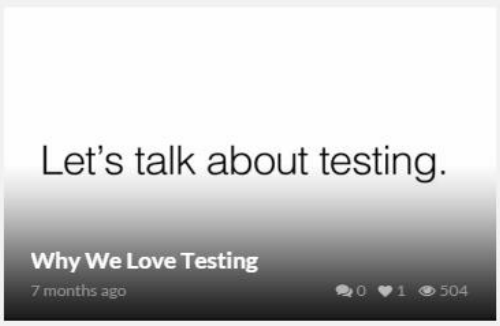
Got tired of reading? No problem! Start watching awesome testing videos…

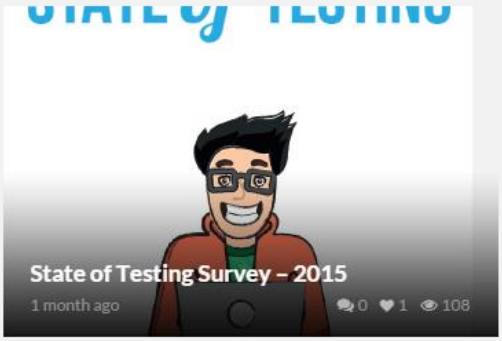
TV for Testers

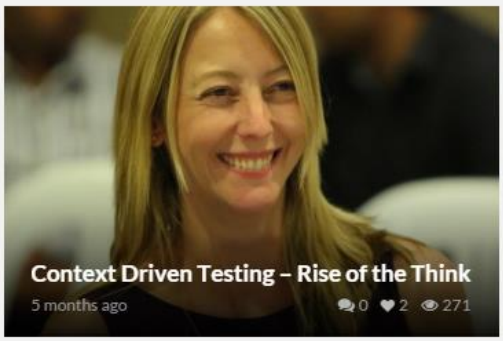
Your one stop shop for all software testing videos

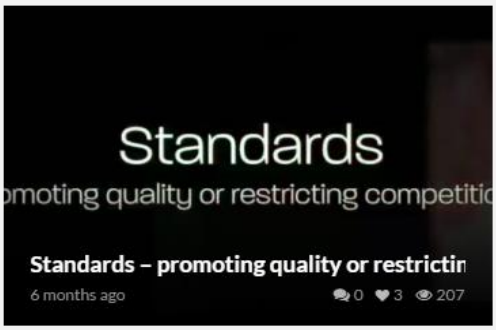












WWW.TVFORTESTERS.COM

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.

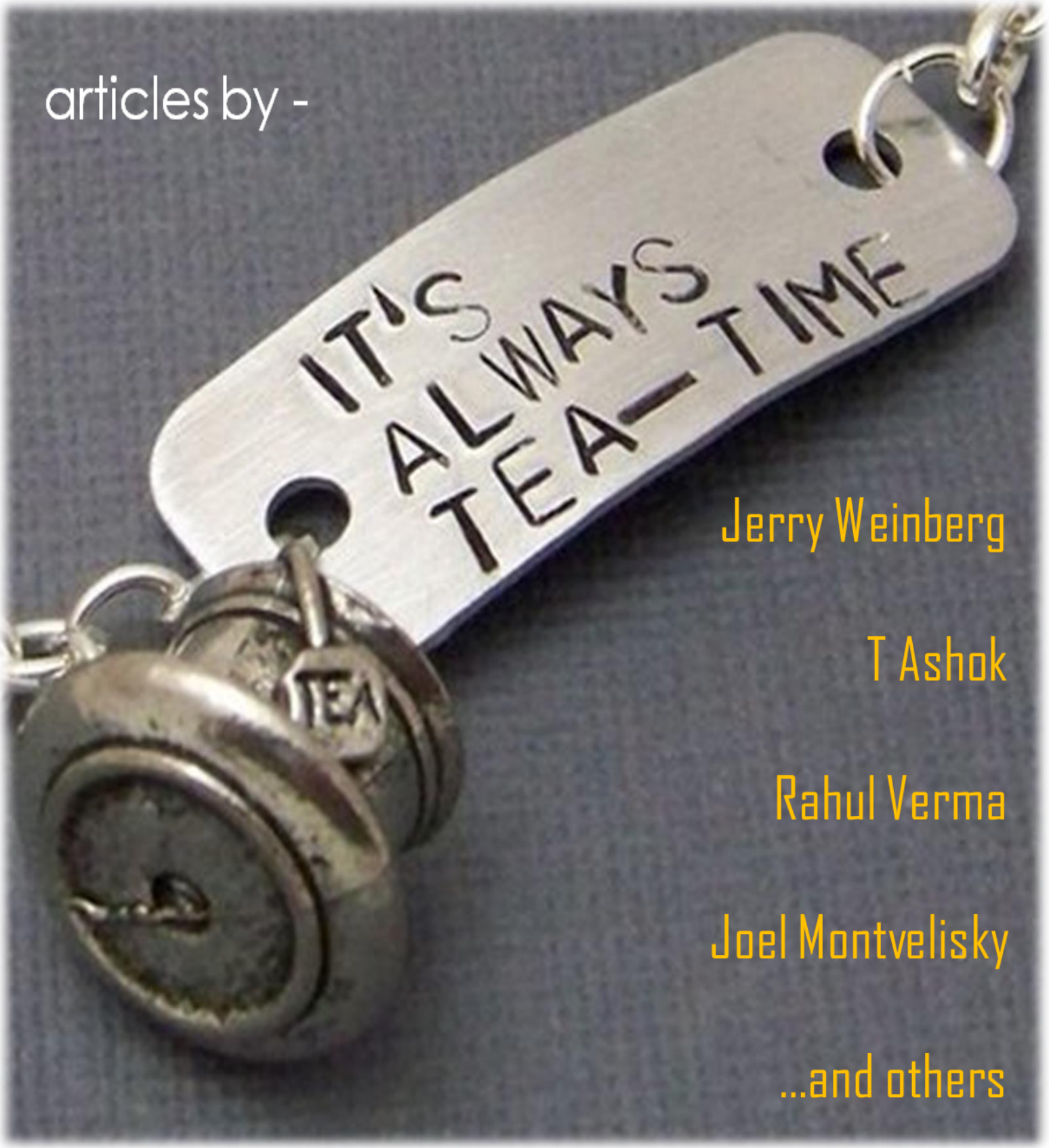
And the **Good News** is…

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 next issue



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 – Twameka Mujumdar

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 mailing list.

SUBSCRIBE

Join our community on

facebook

Follow us on - @TtimewidTesters

www.teatimewithtesters.com