THE INTERNATIONAL MONTHLY FOR NEXT GENERATION TESTERS

Tea-time with Testers

May June 2016 | YEAR 6 ISSUE IV

Articles by –

Jerry Weinberg

John Stevenson

T Ashok

Maaike Brinkhof

Doron Bar

Bas Dijkstra

David Levitt

Over a Cup of Tea with Mike Kelly

AGILE TESTING DAYS

EUROPE´S GREATEST AGILE SOFTWARE TESTING EVENT!

DECEMBER 05–09, 2016, POTSDAM, GERMANY

BEST AGILE RELATED CONFERNCE IN EUROPE
2ND BEST IN THE WORLD

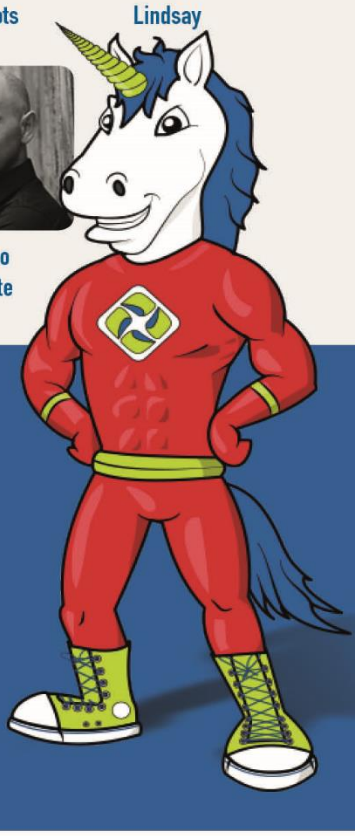
MORE THAN 40 NATIONS

1 UNCONFERENCE DAY, 2 TUTORIAL DAYS, 3 CONFERENCE DAYS

11 KEYNOTES, 12 TUTORIALS, 100 SESSIONS OF TALKS AND WORKSHOPS, LOTS OF BONUS SESSIONS & DAILY SOCIAL EVENTS

THIS YEARS KEYNOTE SPEAKERS

Abby Fichtner	Alexandra Schladebeck	Bart Knaack	Diana Larsen	Gojko Adzic	Huib Schoots	James Lindsay
Jessica DeVita	Keith Klain	Melissa Perri	Michael Wansley	Mike Sutton	Vasco Duarte	



Save yourself some money and get your ticket with 10% off. Just enter following code in to your registration under: **www.agiletestingdays.com/registration**

Discount-Code:

TeaTimeTester_010

For more infos, visit **www.agiletestingdays.com**

TEA-TIME WITH TESTERS

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

Created and Published by:

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

Editorial and Advertising Enquiries:

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

Lalit: (+49) 15227220745
Pratik: (+49) 15215673149

Editorial

A tribute to our Gurus

Guru Purnima was recently celebrated in India. It's a day we celebrate to show respect and gratitude towards our Gurus.

We would like to dedicate this issue to all Gurus and teachers who have contributed to the craft of testing. Before I jump on the next thing I wish to talk about, I would like to mention that I have great respect for all prominent members and leaders from different schools of thoughts and the work they have done in software testing field. But whatever has happened over social media recently and some direct / indirect attempts to show 'Context Driven Testing Community' in poor light has been disheartening and demotivating, not just for me but many other passionate testers who are eager to learn about the craft and taking testing forward.

My own context driven awakening happened when I got in touch with James Bach, Michael Bolton and from there I eventually got introduced to the work done by Dr. Cem Kaner, Jerry Weinberg, Bret Pettichord, Jim Holmes, Matt Heusser and many others. Over these years, I have learned something or the other from my awesome Gurus, colleagues and peers in community.

When it comes to 'challenge each other and learn from there' philosophy of CDT testers, I don't see that as a problem. I personally feel that in absence of this 'challenge and evolve' culture, testing community would either have turned into community of the 'certifieds' or entirely 'checkers' (instead of testers too) by now. CDT members have their own share in helping to maintain 'thinking' part in testing. As a matter of fact, I started to introspect when I was challenged and that helped me critically think about my own beliefs around testing, which facilitated my further learning and improving my skills. I believe most of the testers who are not afraid of learning from mistakes and see things with open thinking pattern would agree to the same.

Of course, we all are entitled to have opinions and freedom of expression. But it is not going to solve the problem if the criticism does not have a defined direction and clarity. And it becomes altogether different thing when that boils down to followers, fans and students of the craft. All I want to say is that the evolution would be hampered if ideas are not challenged. At the same time, I won't hold it against the person if he has high conviction for his beliefs, definitely not when his contribution to the craft cannot be ignored. In the end what community needs most is the SMARTs of all the contributors.

I am a student of the testing craft. Whether it means being pupil of Dr. Cem Kaner, learning from Michael Bolton, referring to Jerry Weinberg's work, going to Jim Holmes for automation wisdom, learning IoT from Paul Gerrard, lessons in leadership from Keith Klain, going to Matt Heusser for Agile / LSD problems or learning buccaneering from James Bach for the betterment of testing field.

Where is the problem? What is the argument? It is all about learning and making an effort to evolve the craft forward, is it not?

Sincerely Yours,

Bhamare -

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

QuickLook

LST News

AGILE TESTING DAYS

EUROPE'S GREATEST AGILE SOFTWARE TESTING EVENT!

December 05–09, 2016

Potsdam, Germany

DISCOUNT-CODE TeaTimeTester_010

What's making News?

Test Masters Academy announces New Testing Conference and Masterclasses - Reinventing Testers Week.

Reported by **Anna Royzman** Program Adviser: **James Bach**

What's in the name? The concept of NEW.

We will look at the "testing as we know it" in a novel, creative way.
We will discover things that are NEW in testing: tools, roles, and interactions.
We will explore horizons and limits of testing: things that have not been considered testing, and now are.
How a tester may move through the role boundaries and how it helps to become a better tester.

We will do it all in a novel conference format that will provide deep, enhanced learning through participation.

Our Methods

We want to take away the fear – fear of the future, fear of authority, fear of misunderstanding, fear of controversy, and fear of change. This conference will offer the innovative personal experience to amplify learning -- through allowing for practical implementation of every topic presented. We will have sessions that demonstrate ideas or practices and allow participants to try them out, the 360* workshops (an innovative experiential technique for deep learning), the Full Spectrum debates where the audience will take active participation, and more.

Your prior conference experiences might have been a "hit or miss". We will prove to you that this conference is different.

Join us for the two unforgettable days that will change your professional career!

Also, in Reinventing Testers Week:

September 25th: Peer **Workshop in Testing Software** (WITS). Attendance is by application or invitation, and limited to 20 participants. WITS is facilitated in the style of LAWST workshops. Presentations are based on the first-person experience of invited delegates. WITS is a peer workshop for practitioners to share experiences in software testing, to allow people interested in these topics to network with their peers, and to help build a community of professionals with common interests. This year, WITS is dedicated to Reinventing Testers theme. To apply, send a request to info @ TestMastersAcademy.org

September 29th: **Quality Leader Bootcamp**. This course is intended for QA and Test Leads and Managers, Test Practice Leads, Test Coaches, Senior Testers, aspiring leaders and everyone who is tasked with establishing professional quality and testing practice in their organizations. This practical course is addressing the needs and skills of QA and Test Leaders and Managers in current workplace. The course material has applied the most recent research and development in Quality and Testing practices and is adapted to the emerging industry trends.

The Workshop In Testing Software (WITS) is the peer workshop for Software Test and QA professionals. Attendance is by application or invitation, and limited to 20 participants. WITS is facilitated in the style of LAWST workshops. Presentations are based on the first-person experience of invited delegates. WITS is a peer workshop for practitioners to share experiences in software testing, to allow people interested in these topics to network with their peers, and to help build a community of professionals with common interests. This year, WITS is dedicated to *Reinventing Testers* theme. To apply, send a request to info@testMastersAcademy.org

INTERVIEW **PART I**

In the community of thinking testers, Mike Kelly is a familiar name. He is the creator of FCC CUTS VIDS heuristic, popularly known as Touring Heuristic.

Currently, Mike is looking after DeveloperTown as its Managing Partner among other interesting things he does.

Interviewing Mike has been on our wish-list and I'm glad that we could finally do it. Special thanks to Dirk for pairing up with me on this task.

Mike has provided insightful answers to our questions and for the benefit of our readers, we are publishing his interview in two parts rather than editing/shortening the answers to fit in one single issue.

I'm sure that you'll like what we discussed.

- Lalitkumar Bhamare

Over A Cup of Tea with Mike Kelly

It's an honor talking with you today, Mike. We would like to know about your testing journey in short.

Thank you. I'm humbled that you'd ask to interview me. I don't do a lot of hands on software testing work any longer, but still view my time developing the skills of a software tester as critical to my success.

In total, I worked as a software-testing consultant for around ten years. In that time I contributed to two books, wrote hundreds of articles, spoke at numerous conferences, and helped launch the Association for Software Testing. My passions were always for test automation, performance testing, and exploratory testing. And early in my career I was blessed with access and friendships with the brightest minds in software testing - particularly James Bach, Cem Kaner, Rob Sabourin, Scott Barber, and everyone else in the LAWST community.

Today I'm the managing partner at DeveloperTown – a software-consulting firm. I'm the founder of Tenant Tracker – a commercial real estate SaaS product. I'm an investor and advisor in numerous startups. And on weekends I'm hobby farmer.

Among other cool things, your name is associated with FCC CUTS VIDS, popularly known as Touring Heuristic. We are curious to know if there is any story behind finding it.

FCC CUTS VIDS emerged out of a weekend spent with James Bach. James was trying to get me to change the way I thought about dissecting a product. He had mentioned the idea of touring a product several times that weekend, and had challenged me with numerous exercises around different ways to identify meaningful dimensions, factors, systems, or components in a product. It was shortly after that weekend that I started to write down my ideas based on a lot of the material we had covered. I view FCC CUTS VIDS as an extension to his work.

After coming up with the heuristic, I used it all the time. It became my default way to start my test planning process for an application or feature. And early in a project it allowed me to quickly identify where I had questions around the various aspects of test coverage and risk.

Considering changing technologies and complexities of software in general, what would you like to add in existing considerations under FCC CUTS VIDS?

Hah. I wouldn't add anything, because then I wouldn't be able to remember it as easily. That's the downside of mnemonics – once you've memorized them they don't like to change.

Based on my more recent experiences, I think I'd try to figure out a way to add two things:

1) Money Tour (or Value Tour): How does this product make money? Or – said in another more general way – how does it deliver value back to the people who produced it? Can you quickly identify all the economic factors either explicit or implicit to the software and the underlying business model behind it? Based on how the product is monetized (or however value is captured), what are the necessary conditions for that transaction (or transactions) to take place? How and when does a user complete a financial transaction? I think this tour helps get at business risk in a way none of the existing tours address.

2) Operations Tour: I'm not going to explain this one well, but I'll try. There are a handful of common technologies/solutions that we integrated with products today that interact with our customers. Examples include Zendesk, Olark, and Okta – simple and focused solutions that touch our customer and their experience with our product(s). There are also a handful of products that we use behind the scenes to monitor and troubleshoot real-time customer issues like New Relic, Google Analytics, or Slunk. They don't "touch" the customer, but we use them to craft better experiences over time. I've come to rely on data from these types of operations-focused tools to help me build my model of what I'm testing and where risk might be. And it also tells me something if we aren't using tools like these to manage our user experience.

You wrote a chapter in "How to Reduce the Cost of Software Testing" which is popular book in testing circle. How was your experience?

My experience working on the book was great, but I feel like I cheated a bit. I was able to repurpose some materials from a series of articles I had written on exploratory testing. So while I refreshed the material and changed it a bit to fit the theme of the book, I had a solid down payment on a first draft before I even started. The editorial team for the book was easy to work with and clear on deadlines. And the reviewers for my chapter were gentile and insightful. I'd do it again without hesitation.

Currently you are focused full time on DeveloperTown. We are curious to know about your experience with DT about building quality in.

Over the last six years we've been working with clients (both single person startups and Fortune 100 companies) to launch brand new SaaS and mobile products. During that period, we also ate our own dog-food and founded our own SaaS startup (www.tenanttracker.net). As a general rule, at DT and at TT, we work in two-week iterations and try to release software as quickly as we possibly can post iteration. Most teams are somewhere between three and ten people in size, and have some mix of design, development, testing, project management, and marketing. Most of us often end up doing some amount of multi-project multitasking despite our best efforts.

On most projects we write "less than perfect" stories and instead try to rely on frequent communication and a high-level of ownership among team members to work through it. Sometimes we have great product owners (clients) who are laser focused and knowledgeable about the product and the market. Sometimes our product owners (clients) are… less than awesome. All of our clients are trying to hit crazy deadlines with about 50% of the money they actually need to do it "the right way."

(If you're one of our clients, clearly you're in the awesome category. I'm not talking about you above.)

We aren't always perfect with standups. We don't always get our retrospectives done. And (depending on the client) some of our planning sessions are more collaborative than others. We have a guiding process, but trust the team to tailor that process on a project-by-project basis.

In our context, testers have multiple projects, crazy deadlines, little documentation, and a lot of responsibility. They also have a lot of trust, we allow them the freedom to select the best tools for the job, and they have a team that (all things being equal) wants to support them as best they can.

We accept that while testing is a role (and it some cases a specialty), we also want it to be everyone's job: our project managers test, most of the time our developers test, most of the time our product owners test, and when we're lucky we can have early alpha/beta customers test. When we assign a tester to a project, we're typically looking for them to find the things that others will miss. We're looking for them to prioritize work based on risk, and to try to identify areas of coverage that other people on the project just aren't thinking of.

We need team members who:

- have great technical skills;
- are comfortable making decisions will little information;
- know how to ask great questions;
- take ownership of their work;
- don't have big egos;
- and are good at communicating with the rest of team.

New product launches – more than any other type of project – best illustrate the tradeoffs of product/market fit, market timing, budget, and technical complexity. Quality in an environment like this is delicate to balance. You're always trying to make the right tradeoff. Even for well-funded large corporate clients, we have to balance tradeoffs. Testers should be some of the best people on the team at identifying tradeoff opportunities and bringing those to the attention of the rest of the team.

You have rich experience with Agile development methodology. From your experience, what trends do you see coming in there that testers must prepare themselves for?

Agile can mean a lot of different things. At risk of avoiding the question, I want to point people back to my answer to the last question. If that sounds like agile to you, then we're doing agile. If not, then we're doing something else and you can ignore this answer.

That said, I've done testing in waterfall environments, RUP, agile, and complete chaos. I've never found that the core skills I've needed to be an effective tester on any of those projects has changed based on the process the team was using. Documentation was more important in waterfall, knowledge of the Rational tools and templates helped with RUP, and the ability to task out your work and estimate quickly will help you in agile – but fundamentally the testing is the same.

For me, test planning is about modeling, assessing risk, identifying coverage criteria. Testing is about designing and executing actual tests based on your models. And test reporting is about effectively communicating those results. None of those core tasks change based on the team's process. What changes are: how much time you have, how much documentation you need, and people's preferences for communication styles and tools?

For me, the trends that matter for software testers working on agile teams have less to do with testing, and more to do with how they can help the rest of the team as a generalist. I think agile testers are expected to be a bit more of a Swiss army knife. Not only do they need to be good testers, but also they need to be amateur developers, have exposure to analytics, need to know the basics of good UX, and need to be excellent researchers. On most agile teams I've worked on as a tester, we've been not only the "tester," but also the backstop for a number of other roles for the project. I think that's a natural extension of our broad focus as testers.

Do you think there will be any scope/demand for specialization?

I always think there will be space for a specialization. But I think what that specialist brings to the table will change over time, and based on the project.

Doing a lot of mobile apps? Your tester is going to have a ton of hardware. Doing an IoT project? Your tester will likely have some hardware/device background to draw on. Building a big data solution? Your tester will likely be able to rock some specialized mathematics, and will likely have some non-trivial performance testing experience etc.

I think that's the challenge. There's an expectation that testers need to simultaneously become generalists and specialists. They need to be able (and willing) to ask "How can I help?" While at the same time bringing some deep experience/skill to the team that others rely on. "I was running some battery-life tests, and it seems like we're still chewing up the battery on the device. Are we properly doing the handoff to the hardware h.264 codec? We hit that on a past project."

While I think that's the trend, I think first and foremost you just need to be a good tester. Being a good tester comes before everything else. If you can't do the core job, you can't paper over it just by being helpful or by having a hard-to-find one-off skill. Agile teams move too fast for underperformers to hide for long. Become a great tester first. Then worry about what else you can bring to the team. The trends will change.

That was Mike Kelly on his past and present work, his opinions about Agile. Join us in part two of this series where we continue to discuss some interesting aspects about tester's skills, other fields that fascinate Mike and some other cool topics.

Stay tuned for part two….

- *Lalit and Dirk*

Why Agile Projects Sometimes Fail

The gang was enjoying a BBQ Pig Out at Rudy's. It was a magical moment until Rusty and Millie started to argue about Agile software development.

Rusty started it all by saying, "Agile is magical."

Millie banged on the table with a half-chewed pork rib. "That's ridiculous. There's nothing magical about it."

"Sure there is." Rusty pulled a Sharpie out of his pocket protector and printed "AGILE" on a paper towel (which passes for a napkin in Rudy's). "There are just a few things management has to provide—like MONEY." He sketched a capital M on the towel, making MAGILE.

"Money's not enough," said Millie.

"Of course not. Management has to eliminate environmental interference.' With one smooth stroke, he crossed out the "E."

Millie frowned and shook her head, but Rusty took no notice. "And they need to Cooperate, and not just occasionally, but All the time." He added the C and A, finally producing "MAGICAL."

"Cute," said Millie, her tone sarcastic, but she was clearing struggling not to smile. "But successful projects require more than waving a Sharpie wand and pronouncing 'AgileCadabra.'"

We all knew that Rusty was pulling our legs. Millie, of course, was right. If you want to succeed with an Agile approach, you need more than magic rituals. Not only that, you need to avoid quite a few rather common mistakes that lead to failure.

Common Mistakes in Building New Things

In my experience, these common mistakes are not unique to Agile projects, but will kill Agile projects just as easily as they kill Waterfall or any other approach:

1. Committing to a schedule or cost without having any relevant experience with this type of project.

2. Using the experience on a similar but smaller project to commit to an estimate on a larger project.

3. Extending requirements to "optimize" or beat unknown competition.

4. Failing to recognize signs of impending failure and/or act on them to extend schedules, reduce costly requirements. (like those that diminish velocity by creating more frequent failed tests).

5. Failing to recognize limits of the environment or process or recognizing them but being unwilling to change them.

6. Simply undertaking too many simultaneous tasks and perhaps failing to complete any of them.

7. Not recognizing both changes and opportunities presented by a new technology.

8. Not asking the customer, out of fear, or lack of customer surrogate contact.

9. Not asking anyone for help (fear?).

10. I invite my readers to contribute more failure dangers to this list.

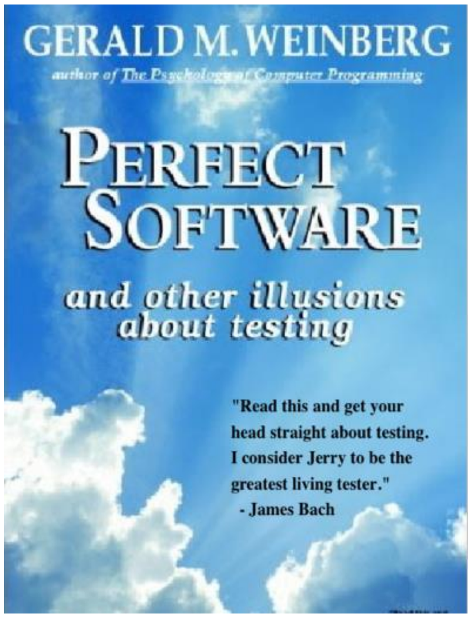
The Underlying Failure

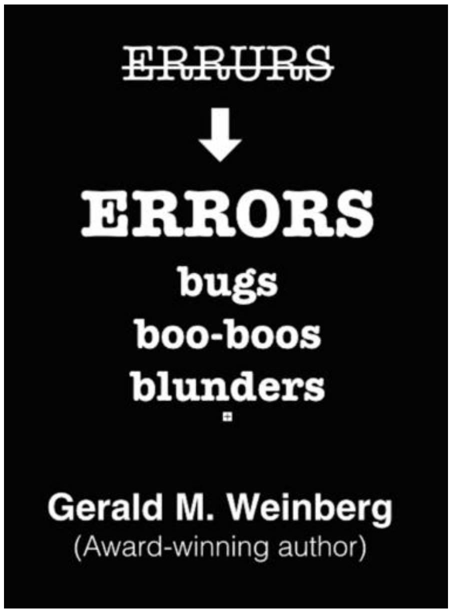
Beneath each of these failure reasons, and others, lies one generalized failure. I explain that failure in the remainder of this article, posted as a chapter in my book, Agile Impressions.

Back To Index

HIGHLY RECOMMENDED

If you want to learn more about testing and quality, take a look at some of Jerry's books, such as:


GERALD M. WEINBERG
author of The Psychology of Computer Programming
PERFECT SOFTWARE
and other illusions about testing
"Read this and get your head straight about testing. I consider Jerry to be the greatest living tester." - James Bach



People Skills—Soft but Difficult
ARE YOUR LIGHTS ON
Donald C. Gause
Gerald M. Weinberg
What Did You Say?
The Art of Giving and Receiving FEEDBACK
Exploring Requirements 1: Quality Before Design
Exploring Requirements 2: First Steps to Design
BECOMING A CHANGE ARTIST
MORE SECRETS OF CONSULTING
Becoming a Technical Leader
Gerald M. Weinberg

Curious to know why you should get 'People Skills' bundle by Jerry? Then check this out

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

Jerry Weinberg has been observing software development for more than 50 years.

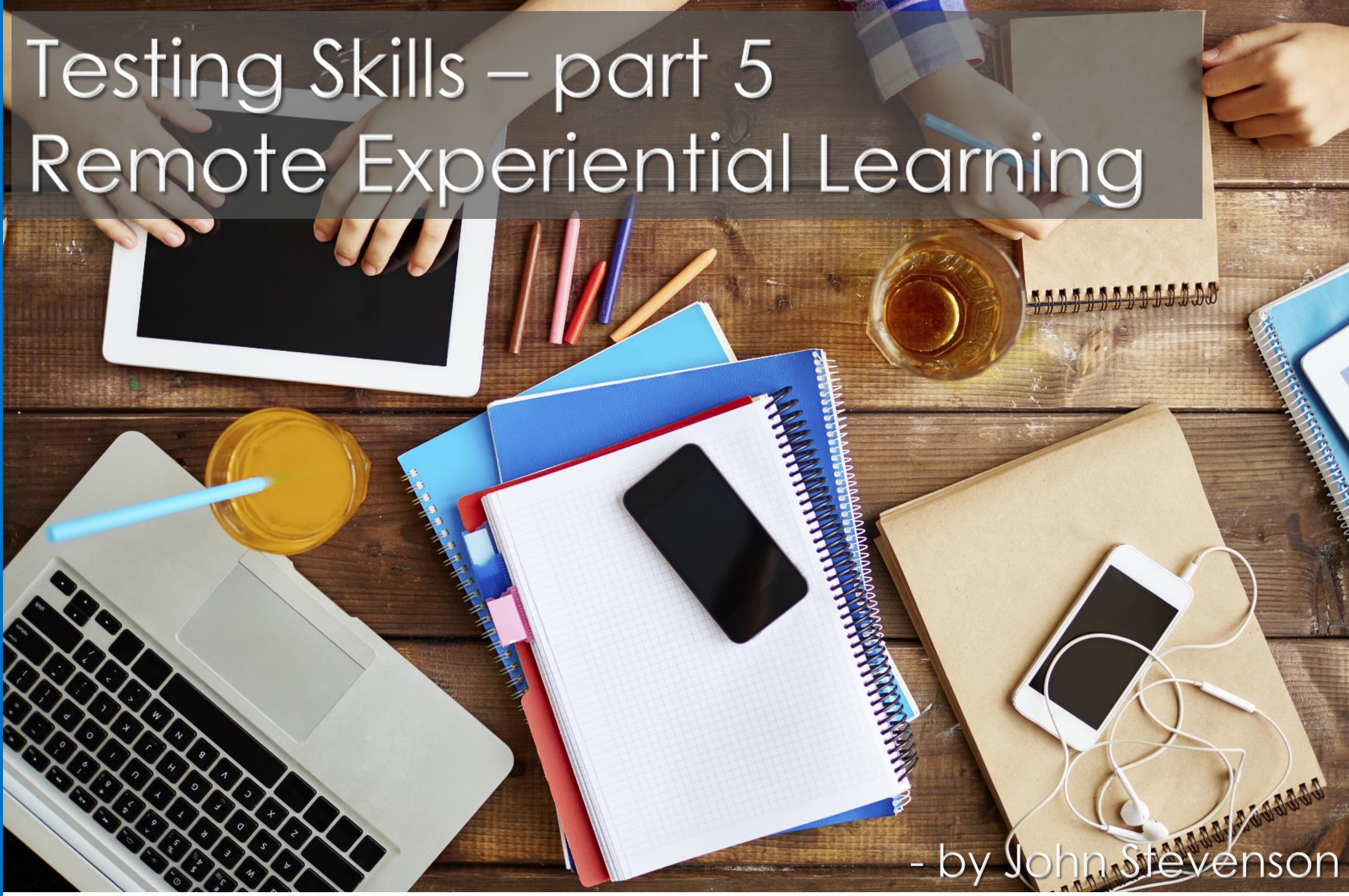
Lately, he's been observing the Agile movement, and he's offering an evolving set of impressions of where it came from, where it is now, and where it's going. If you are doing things Agile way or are curious about it, this book is for you.

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

TTWT Rating: ★★★★★

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

Testing Skills – part 5 Remote Experiential Learning



- by John Stevenson

Many people work with teams which are globally distributed, this has some logistical issues, one being how to implement useful and practical training approaches. One common approach used is C.B.T. (Computer Based Training). This is where participants login in and listen to pre-prepared exercises and videos, sometimes with a test at the end. Another approach is to arrange a video session with an online tutor where they go through the material and the participants can ask questions whilst listening to the tutor. These are OK as learning tools, but it is difficult for the participants to apply the knowledge learnt to their daily role.

There is an alternative distance learning approach that I experienced whilst attending an online workshop run by The Growing Agile team (Samantha Laing and Karen Greaves). I have since this course created my own remote workshop using this approach with some success. What follows is an introduction to this approach. Hopefully you can take this and adapt it for your own teams.

The basic principles of this remote training approach is based upon the 4Cs as described in the book "Training from the back of the room" by Sharon Bowman. Each of your learning elements should include all elements of the 4Cs in each module.

For each module of the course I create a workbook which goes through each aspect of the 4Cs

The first 'C' is Connect

Before you start teaching the students ask them what they already know about the topic. Create activities they can do offline to find out how the topic is relevant to their current role or what they currently know about the topic.

The next 'C' is Concepts

This is the traditional learning part, where you can introduce and explain what the topic is about. You can do this as either a series of written articles or pre-recorded videos.

The third 'C' is Concrete practice

Students apply the concepts in practice. If you are running this remotely you can set up activities and exercises related to the concepts which the students should, ideally, apply to their own working domain.

The final 'C' is Conclusions

This is best to done as a small group, maybe as an online video call. All the students get together and discuss what they have learnt. This is a great way to reinforce the learning since each person should bring different examples of applying the learning to the discussion and provide a more context rich learning experience.

When you are looking to create any remote learning experiences it is worthwhile making sure that each of your training sessions covers all aspects of the 4Cs. An advantage this learning approach gives is that it requires only a couple of hours of learning from each participant. They can do this at their own pace and then discuss their learning and how it applied to them during a weekly hour long video conference call with the others taking part in the course. It is crucial to set your expectations of the participants and get them to give a commitment to spending some time doing the exercises before the video call.

As an additional option when I ran my remote workshops I set up a closed wiki site so that the participants could have discussions and provide some information about what they have learnt. Also with permission from the participants I recorded the video sessions and uploaded them to the wiki so they could go back and watch them later.

John is tester, blogger, tweeter and author who has a passion for the software testing profession. He is keen to see what can be of benefit to software testing from outside the traditional channels and likes to explore different domains and see if there is anything that can be of value to testing. At the same time he likes to understand the connections between other crafts such as anthropology, ethnographic research, design thinking and cognitive science and software testing. He is currently writing a book on this called "The Psychology of software Testing".

John has presented workshops and presentations at various events such as Agile Alliance, CAST, Testbash and Let's Test.

Mindfulness and Exploratory Testing

- by Doron Bar

Introduction

Many words, articles, and books have been written about Exploratory Testing: definitions, ways of implementation, examples and more. However, I am not sure that someone tried to define or recommend the mental state which we should adopt when we perform these tests.

To this end, I would like to introduce a form of meditation that comes originally from Buddhism, and is called "Mindfulness". This psychological condition also can be applied not only on dedicated time but also while we experience life in both the professional and personal paths.

I know that we are part of the technological world, and here I am bringing a doctrine dating back thousands of years. Well, the world may have changed, but humans - not yet.

What is Mindfulness?

Mindfulness is consciously paying attention to the present. It is to be with yourself (thoughts, actions, emotions, physical sensations) in a non-judgmental manner. Be tended with what is happening now, with this special moment.

For example, when we talk with someone in a mindfulness state, we listen to what he says, his words choosing, his body language (even if his/her words are harsh). We are aware of what is happening during the conversation, how that influence us.

Why is this important?

Our focus at present is reinforced when putting the future concerns or fears from the past to the present. But we can find it easier when we are able to separate the experience of its jurisdiction and hence to make smarter decisions.

How to enter this State?

We observe. We suspend judgment and response. Just be in the moment, let only to your awareness to work, not to your thoughts.

If something like sounds or thoughts, related or not, try to disturb us, we will be aware of it and gently return to the situation of mindfulness. We need to remain in full awareness to get the most out of this experience.

We will be open to data from the senses.

We describe what is happening sharply and without a mixture of interpretation.

We participate in the present when we do not bring other emotional baggage. We will exhaust the experience to the fullest.

To describe the process more precisely, and in an easy to remember manner, I found this acronym:

STOP

Stop. Something happens that you want to deeply understand? Did you flood with emotions? Stop for a moment before acting or even interpreting the incidence.

Take a breath. Inhale and exhale: concentrate on the aspiration of air that is filling our lungs, on the exhaling.

Observe your experience. Describe what happened in a factual form, describe the feelings, the thoughts.

Proceed. Proceed with the insights you received.

But what does the mindfulness have to do with Exploratory Testing?

Let’s say we have a product ready to be explored, environment set, and understanding of the product. We wrote the charter for testing or deciding what kind of testing we will do, what we will focus on and so forth. In short, can we start?

Not before we confirm that we have this psychological platform. Exploratory Testing is an activity that - we always knew- requires the full resources of the tester, his full commitment. Eventually, the tester needs to learn about the software, plan the next test steps and conduct the testing, all the same activity. Unlike scripted testing, where you can stop almost anywhere, prepare coffee and return to the next step, here the tester should be at her or his peak concentration to succeed.

To achieve this, I suggest using Mindfulness.

This basically means we are entering a situation where we are consciously paying attention to the present, it is to be with yourself (in your thoughts, actions, emotions, physical sensations) and the product in a non-judgmental manner. Be present in the fullness with what is happening now, with this special moment. We will not accept to interfere with our random thoughts or concerns. We do not bring old feelings to the process ("there is no way it will work this time", "this has not been defined, AGAIN", "the developer irritates me"). We'll have no concerns about the future ("will they appreciate my work?" "Will the product be successful?"). Just allow your awareness to interact with the product, test it, see it, explore it, get insights about it. You should also during operation describe the flow, steps you performed, you can register the feelings you experienced (but as outer experience, not to give to these feeling).

Awareness relates directly to the experience without the mediation of thinking. There is no difference between inner and outer, between the examiner and examinee, between being and doing. The ego is not experiencing with what happens. It is to notice and at the same time be aware of what happens in a non-conceptual manner.

At any point during the tests, especially after the completion of testing of the hypothesis we made during the test, you can (and should) stop. Take a few breath of air, observe and describe what we did and how the program is responding, now you can think where we go next and continue the process: take what we experienced in the previous steps, directing the continued testing.

We might be disturbed during the tests (although it is better to pre-arrange a quiet surrounding, always something unexpected might crop up). Maybe other concern might raise, perhaps even good memories. We must always be mindful, and when that happens, do not be angry at ourselves, but to move these distractions aside.

While the actions we perform, we are mindful, not thinking. We are very aware of what is happening, we perform the steps we planned, we perceive with their senses what is happening in front of us, the behavior of the software. We note what we felt but as mere onlookers, without identifying with the emotions at this time. Stop, describe (to ourselves or in writing), plan and continue. Mind you that in any case, we don’t really “Plan, test and learn” at the exact same time, but rather in the same activity.

Stuck without ideas, something happens that is not clear, or something is clear but we do not understand how we got there, and other similar situations, we will remain in control and not subdue to anger, or by concerns that perhaps we do not understand what is happening or perhaps we are not good enough. We will not run the developer. We will just stop, breath; we will describe what happened, and we'll try and calm and in awareness try to figure the sources of the problem, consider the requirements, what we know about the product, the technology, and then we try to understand the source of what we saw. Did we not find it? Well okay, we'll list it and investigate later.

By the way, you can implement this change even when the manager tells us something that sounds abusive, or when your child does something that seems to us as offensive. Do not give ourselves up to our emotions. Stop, breathe, describe what happened, try to understand what is said or done, think about the reasons why that happened and how to act, and continue.

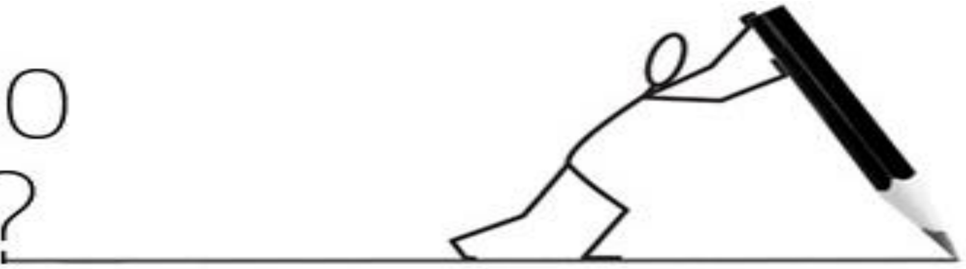
Epilogue

Now, after reading, I hope this has given you some understanding on the subject of mindfulness. This is the best way I know to carry out such tests, and maybe also the way to treat the world. In addition, I hope it will give you the motivation to read more about mindfulness.

But most importantly, try the above in the next tests you are going to perform!

Doron has filled many software testing positions since 1998. Among his various positions, Doron was a software tester, team leader, testing manager and testing project manager, both in startup companies and in leading corporations. In his current position at AVG, Doron serves as a testing architect in an Agile environment. This position calls for analyzing requirements and writing the test scope, determining standards and processes, identifying and implementing advanced methodologies and conducting the end game.

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

Mapping Biases to Testing - part 2!

- by Maaike Brinkhof

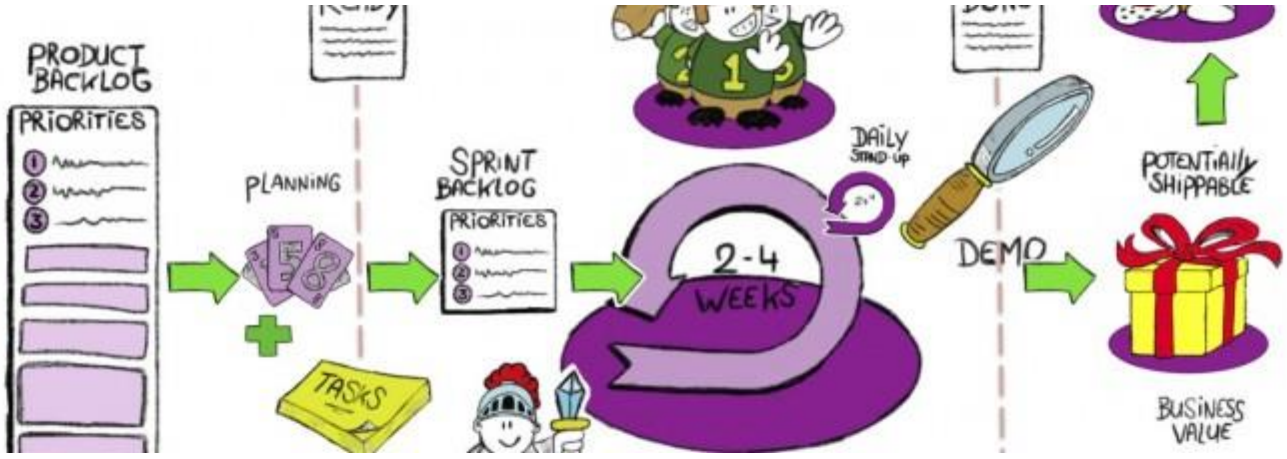
Dear reader, welcome back to the Mapping Biases to Testing series. Today it is my pleasure to discuss the first bias in this series: the Anchoring Effect. Before we start mapping that to testing, I want to make sure that we have a clear understanding of what the anchoring effect is.

“Anchoring is a cognitive bias that describes the common human tendency to rely too heavily on the first piece of information offered (the "anchor") when making decisions. During **decision making**, anchoring occurs when individuals use an **initial piece of information** to make subsequent judgments. Once an anchor is set, other judgments are made by adjusting away from that anchor, and there is a **bias toward interpreting other information around the anchor**. For example, the initial price offered for a used car sets the standard for the rest of the negotiations, so that prices lower than the initial price seem more reasonable even if they are still higher than what the car is really worth.”

I highlighted the important parts. Decision making is something we constantly have to do during testing, and it is important to realise which anchors might affect you. Also, to make this clear, I think ‘testing’ is not just the act of doing a test session, but thinking about everything that involves quality. You can apply a testing mindset to all that is needed to make software: the process, the specifications, the way the team works, etc.

My experience

Personally, some scrum artefacts are anchors for me, namely the duration of the sprint, estimation of stories and counting bugs to measure quality. Let me explain this with examples.

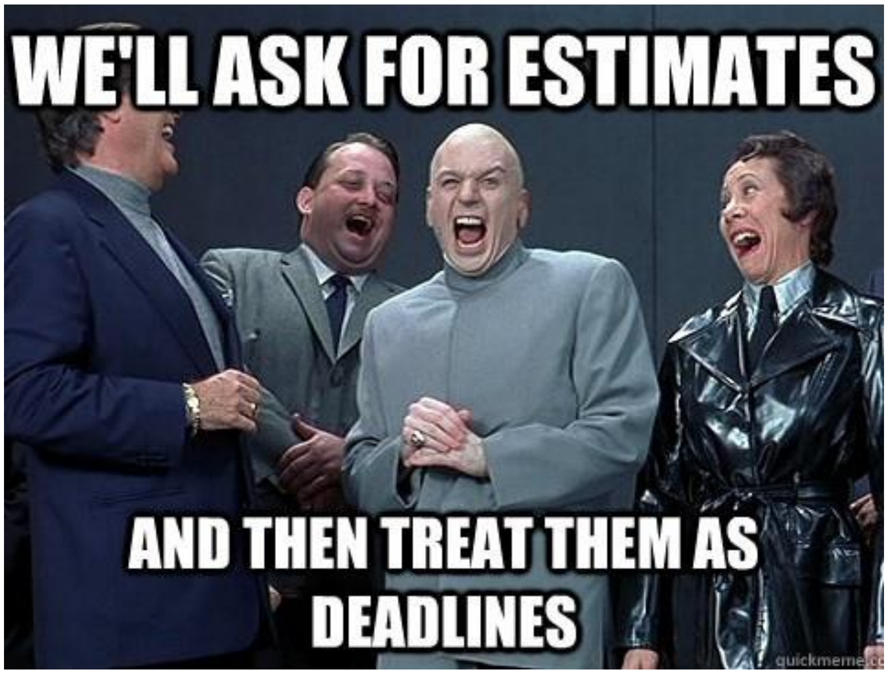


The clients I worked for all had sprints that lasted two to three weeks. Those of you also working with the Scrum framework know the drill: you create a sprint backlog consisting of stories and make sure the work is done at the end of the allotted time. What I have seen happening again and again is the last day of the sprint being a hectic one, with the focus on testing. That's because a lot of companies are secretly doing the 'scrumwaterfall'. Development starts at the beginning of the sprint, but testing is still an activity that takes place at the end. The business wants to get the stories done, so testing is rushed. The duration of the sprint suddenly has become the anchor. It takes a lot of courage as a tester to change this by speaking up, giving options to solve this, and not succumb to the pressure of cheating the Definition of Done.

Sadly, I've witnessed teams cheating the Definition of Done because it was the last day of the sprint and they were under pressure to deliver the work. Low quality work was accepted and the fact that technical debt will come back to haunt the team wasn't a consideration at that moment.

The anchor of the sprint is strong. When you're working with Scrum you are drilled to think in these increments, even when reality is sometimes more obtuse. You could say that the reason stories don't get completed in time (or with low quality) is also because people are very bad at estimating the stories. That brings us to the next anchor.

Estimation of stories



Estimating is something that has fascinated me since I first stepped into the wondrous world of office work. I still wonder why people put so much faith in a planning, why managers judge profit against a fictional target they produced months ago, why people keep getting surprised when a deadline isn't made. Can we really not see that a complicated reality, consisting of so many uncontrollable factors, cannot be estimated?

A movement called 'No Estimates' is on the rise to counter the problems that come from estimating. Personally, I haven't read enough about it to say "this is the solution", but I do sympathise with the arguments. It's worth investigating if this sort of thing interests you.

Something I have witnessed in estimating user stories, is that the estimate is usually too low. The argument is often "yeah, but we did a story similar to this one and that was 8 points". That other story is suddenly the anchor, and if you estimate the new story at 13 points, people want an explanation. I always say: "There are so many unknown factors", or the even less popular argument of "we have a track record of picking up stories that we estimated at 8 points, but didn't manage to finish in one sprint". Sadly, such an argument rarely convinces others, because the belief in estimates is high. I have succumbed to the general consensus more often than I'd like to admit. Trust me, I get no joy from saying "I told you so", when a story that we estimated at 8 points (and I wanted to give it 13 points) ends up not being done in one sprint. I keep my mouth shut at that point, but during the next planning session I will say "remember that 8 point story? Yeah…let's not be so silly this time", and the cycle can repeat itself.

My most ridiculous example comes from a few years back. I worked for a large company back then; let's just say they were pretty big on processes and plans. Every release was managed by at least 10 managers, risk was a very big deal. The way they handled the risk though, with anchors, that was a bit crazy. A new release was considered 'good' if it didn't have more than 2 high severity issues, 5-10 medium severity issues, and any amount of low severity issues. The Defect Report Meetings were a bit surreal. There we were in a room, with a bunch of people, discussing lists of bugs and saying 'the quality is okay' based on numbers of bugs. The amount of time we wasted talking about low severity bugs could probably have been used to fix those. Office craziness at its finest. I hope the anchor is clear here, but let me say it very clearly: The quality of your product is NOT based on the amount of bugs you have found. Taking that as an anchor is making the discussion and definition of 'quality' very easy and narrow, but it's also denying reality. Quality is something very complex, so be very careful to resort to anchors like 'how much defects of type A or type B do we have' and basing your judgement on that alone.

What can we learn from this from a test perspective?

As a tester, you have to act as the conscience of the team. If it is in your power: don't let a bad estimation, or sprint that is in danger of not getting done completely, affect your judgement. Our job is to inform our clients and teammates of risks we see in the product, based on sound metrics and feeling (yes, feelings!). If there was not enough time to test thoroughly, because the team fell for the anti-pattern of Scrumwaterfall, try to take steps to combat this (improve the testability of the product by working more closely together with the developers, for instance).

If you are under pressure from outside the team to deliver the software, even when it is not done yet, make the risks visible! Inform, inform, inform. That should be our main concern. Although*, if my team would constantly be forced to release low quality software, I would get kind of depressed with the working environment. However, sometimes it happens that someone higher up the chain makes a decision to bring live shitty software.*

Also, don't forget to take a look inwards. Are there anchors that are influencing your work? Do you count the bugs you find and do you draw conclusions from there? Do you write a certain amount of automated checks because you think that sounds about right? Are there any other test-related numbers that seem normal to you? If so, challenge yourself to think 'is this normal or could it be an anchor?'

What's next?

In this article series I hope to shed some light on a number of biases and fallacies and what harm or good they can do in testing. I will cover the following biases, fallacies and effects:

- availability heuristic
- framing
- sunk cost fallacy
- cognitive ease
- confirmation bias
- priming
- hindsight bias
- attribution bias
- *"What You See Is All There Is"*

If you have more input for biases or fallacies that you want to see covered, please leave me a tweet @Maaikees

Maaike Brinkhof - Test Consultant at Xebia Netherlands

Maaike is an agile tester. She loves testing because there are so many ways to add value to a team, be it by thinking critically about the product, working on the team dynamics, working to clarify the specs and testability of the product, getting the whole team to test with Exploratory Testing…the options are almost endless! She likes to help teams who are not sure where or what to test.

After reading "Thinking, Fast and Slow" by Danial Kahneman she developed a special interest in biases with regards to testing. During 'analogue time' Maaike likes to practice yoga, go for a run, check out new local beers, play her clarinet and travel with her boyfriend.

Back To Index

Teaching a software testing class

- by David Levitt

I’ve been a part-time Computer Science instructor for over 10 years, but I’ve never seen a course devoted solely to software testing. I haven’t even seen it given much attention at all. It seems odd that such a critical skill is not given the same attention as other areas of study, such as programming, data structures, databases, and the like, so I thought I’d set out to change that. As a fan of Tea-time with Testers, I thought other readers might be interested if I shared my experience.

First, some background. I teach part-time at Metropolitan State University located in St. Paul, Minnesota (USA). We offer Computer Science and related degree programs. Every semester, we like to offer a special topics class. Our department head was very enthusiastic. However, as a matter of protocol, but not necessity, I needed to present the class and its rationale at an upcoming department meeting.

Most instructors were also supportive of the idea. However, one of the more influential full time faculty members said outright it would never fly due to lack of interest. Truth is, there was more accuracy to her prediction than I wanted to admit, which I’ll get too shortly. For the moment though, I countered her comment by saying that years ago, that might have been true. In today’s world though, developers are expected to test their code, and the reality is that most students are not exposed to even the fundamentals of software testing. Needless to say, my point was acknowledged, but I now had to pitch the idea to students.

The reality is that testing is not viewed as one of the “sexier” Computer Science topics, and in fact, my efforts to teach such a class failed several years ago at another college. It was a similar playbook too: the dean approved, faculty approved, but the course got dropped due to low enrollment. This time though, I did some promotion by visiting a few classes and giving a quick 5 minute talk. I called out to students that in today’s environment, employers are looking for “T” resources – deep in one area of expertise, but broad in others. Specifically, testing is a skill that could make you more marketable as a programmer, and it might even open up new doors in software testing. Apparently my visits were effective because enough students enrolled this time. I now had to quickly turn my attention to planning the course.

My overarching goal was to be pragmatic and emphasize real world techniques rather than theory (I did do some theory though). I also wanted to provide a thorough grounding on concepts such as soft skills, static testing, black box testing, white box testing, unit testing, system testing, and test planning. Quite simply, I could not find a single college level text that met my requirements, so I decided on several professional grade books instead. Here's what I chose and why:

- "A Practitioners' Guide to Software Test Design", by Lee Copeland, This is a great little book on a variety of testing techniques. On its own, it's a bit light, so I had to supplement it with supporting details.
- "Lessons Learned in Software Testing - A Context-Driven Approach", by Cem Kaner, James Bach, and Pritchard. Nothing in this book is technical, but it offers a lot of good, practical information on the softer skills of the craft.
- "How to Break Software - A Practical Guide to Testing", by James Whittaker. This book presents a well-organized set of black-box software attacks. Sadly, the software that comes with the book is old and does not work on 64 bit machines. We worked past that though.
- '"Practical Unit Testing with JUnit and Mockito", by Tomek Kaczanowski. Finding a suitable book for white box testing was the hardest to find. Many books on jUnit are either out of date and / or do not contain exercises. This book has both, and it also provides an up-to-date treatment on Mockito – a popular mocking framework.

I also drew material from other sources, most notably "How Google Tests Software" and "How Microsoft Tests Software". All-in-all, I felt I was providing pretty good coverage, but I also wanted to convey that testing is as a dynamic area of study just like any other Computer Science subject, so I supplemented the weekly assignments with an online discussion. Students were to find a short article or video of their choosing from the web (including TTwT) and post a summary on the discussion board. They also had to respond to at least two other students. The articles students found were fabulous, and it exposed students to just how rich and exciting the testing field really is! It generated some great conversations and I, too, learned a great deal along the way. Many may not realize this, but learning is one of the greatest side effects of teaching!

We covered a lot of ground in one semester. We started with a classic testing problem of generating test cases to determine if three numbers are a triangle: http://www.testing-challenges.org/Weinberg-Myers+Triangle+Problem. From there, we covered test case design techniques and lessons learned. We then spent a few weeks attacking our school's student portal called Desire to Learn (D2L). Finally, the last few weeks were spent on jUnit and Mockito.

In retrospect, I found some interesting parallels between my teaching experience and what I see as a practitioner. Case in point: I see a lot of enthusiasm for automated testing, yet many testers lack a good grounding in the fundamentals, like equivalence partitioning, boundary value analysis, path analysis, test data design, and pairwise testing. More interesting is something that in my opinion does not get enough attention, namely static testing, a.k.a. preemptive discovery. When presented with requirements or a problem statement, I have found many students and practitioners lacking in their ability to do it well. It's a shame because finding defects early promotes "fail fast, fast feedback" thinking. In the classroom, I was able to address this by assigning more exercises. It's a different story in the real world though, where tight deadlines, culture, and other factors prevail. My hope is to see more attention paid to static testing in Tea-time with Testers and the like. Time of course will tell.

Side note: As a matter of pragmatism and thoroughness, I had to acknowledge that oftentimes testers don't get testable requirements, if they get any at all. I addressed that reality by spending several weeks on exploratory testing and the structured attacks like the ones promoted in James Whittaker's book.

Student feedback was overall very positive. Several commented that many of the concepts they learned should have been taught in earlier classes. Others requested an advanced testing class, and I could not agree more. There's so much content I just did not have time to cover, such as automated acceptance testing via Cucumber, or web testing via Selenium. I would also like to incorporate a Whittaker-like book that focuses on attacks for cell phones and mobile devices. Finally, I would like to set up a testing lab and provide a full stack of tools and a full lifecycle project students can work on.

One or two students commented they did not like having all of the books, and I would have to agree. Lastly, one student, a professional tester by trade, mentioned he had to do quite a bit of searching to find a testing class. Whether testing is embedded into other courses or taught standalone, I do believe it's time for academia and practitioners to treat testing as a first class citizen just like any other area of study. In any event, that's my opinion. I'd be interested to hear your thoughts, either on the content of the course, suggestions for books, or your own experience as a practitioner or educator.

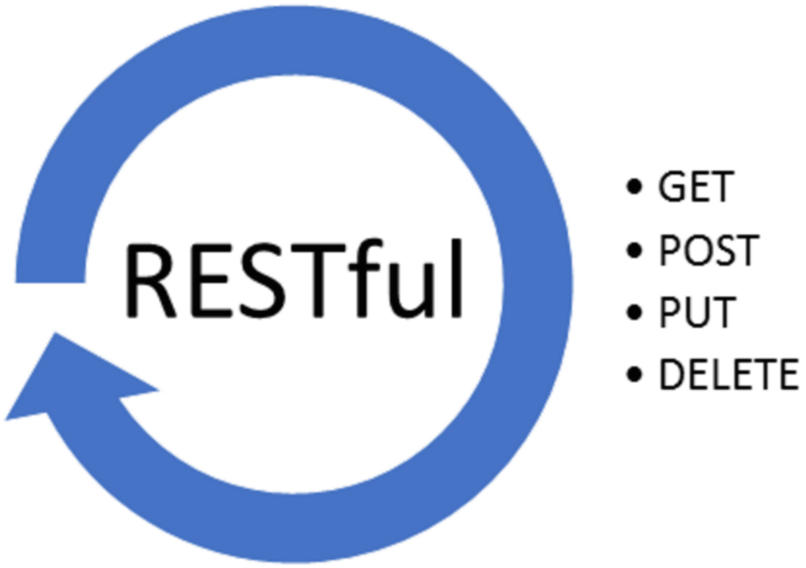
Mr. Levitt has been in the software field for over 35 years.

He is a full time practitioner who also has a passion for teaching. He's held lead roles as a programmer and a tester, and works mostly with large scale systems. He holds a BS and MS in Computer Science and an Advanced Certificate in Software Engineering.

He can be reached at david.levitt@metrostate.edu

Testing RESTful web services with REST Assured

- by Bas Dijkstra



In the last couple of years, a growing share of developers and IT organizations have started to expose their applications and services – or part thereof – to the rest of the world through APIs. These APIs enable other developers to create new solutions that integrate and communicate with these existing systems. Examples of such APIs are for example Google's Gmail developer API or PayPal's developer API

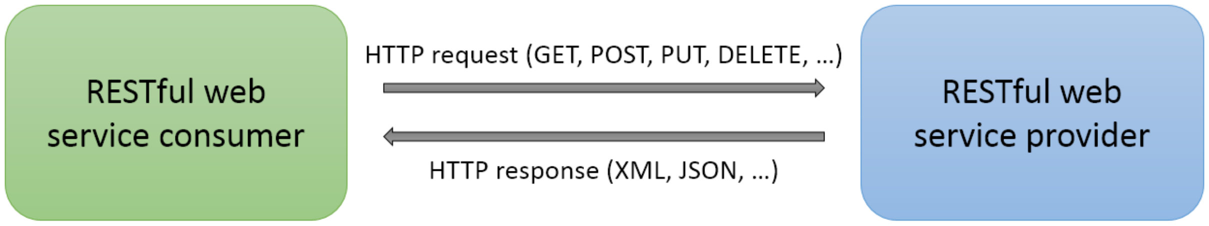
Also, we have been seeing a transformation in modern IT solutions from the traditional monolithic architecture towards architectures based on microservices and self-contained systems. These architectures are characterized by small, independent and reusable services with a single responsibility that are connected to build larger and more complex applications.

Some more trends that will have been observed by anyone with more than a passing interest in software development and IT in general is the steady increase in the number of mobile applications and the fact that we are also starting to see more and more applications for the Internet of Things (IoT).

A common factor in all of the trends mentioned above is that they often rely on RESTful web services to expose the possibility for two or more applications to interact by reading and writing data. But what exactly is a RESTful web service? How can you, as a tester, create and execute tests for these web services? And what tools are available for you to make testing these web services easier? In this article I will answer these questions using an example public API and the REST Assured Java library.

What is a RESTful web service?

Let us first take a look at what exactly a RESTful web service is. In short, a RESTful web service is a web service based on the Representational State Transfer principle. This means that the web service exposes basic HTTP request methods, such as GET, POST, PUT and DELETE, and URIs (URLs are a form of URIs) to allow consumers of that service to perform create, read, update and delete (CRUD) operations on data.


RESTful web service consumer
HTTP request (GET, POST, PUT, DELETE, ...)
HTTP response (XML, JSON, ...)
RESTful web service provider

For example, when you perform a GET request using the URI `http://somehost/users/Tom`, it is expected that data related to the user identified as Tom is returned. This data is typically returned by the web service either in JSON or in XML format, although REST does support many other data formats as well. You can compare performing such a GET request to what your web browser does when you load a web page, since retrieving a web page from a web server is also done by executing a number of HTTP GET requests. You would only have to take a look at the logs generated by tools such as Wireshark or Fiddler to see what I mean.

Comparing RESTful to SOAP-based web services

So, how do these RESTful web services compare to 'traditional' SOAP-based web services? The most important difference between REST and SOAP is the fact that REST is an architectural principle, while SOAP is a standardized protocol. Still, RESTful web services often make use of standards, such as HTTP, JSON or XML.

Here are some key differences between RESTful web services and SOAP web services:

- REST allows for a variety of message formats including, but not limited to JSON and XML, while SOAP exclusively requires the use of XML (with the possibility of other message types wrapped inside the XML request)
- RESTful web services typically have smaller overhead, leading to better performance and scalability – this is the main reason for its popularity in mobile and IoT applications
- So, why is SOAP still being used, then? Some reasons for using SOAP instead of REST could be the support that SOAP provides for:
- WS-Security – RESTful web services can of course be secured, but there is no built in equivalent for WS-Security
- WS-ReliableMessaging – Since REST is not a protocol, there is no built in mechanism to guarantee that a request is actually delivered
- WS-AtomicTransaction – Again, REST is not a protocol and as such it does not feature a mechanism to guarantee atomicity of transactions

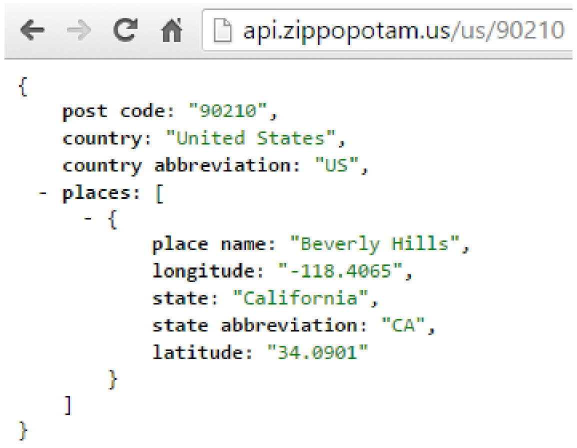
How can you test RESTful web services?

Now that we have seen what RESTful web services are and how they compare to SOAP-based web services, let's take a closer look at the ways we can performs tests on them. Since RESTful web services perform CRUD operations on data, it makes sense to investigate these CRUD operations (next to executing other types of checks, which I will touch upon later on).

For example, to test the lifecycle of data exposed by a RESTful web service we could:

- Perform a HTTP POST-request to `http://somehost/users/` with the required input parameters to create a user `Tom`
- Perform the previously mentioned GET request to `http://somehost/users/Tom` to check whether the user has been created successfully and also to check if all relevant data has been stored correctly
- Perform a PUT request to `http://somehost/users/Tom` to update one or more of the data (for example, when Tom moves house, it makes sense to update his address details)
- Repeat the GET request to check whether the data was successfully updated
- Perform a DELETE request to `http://somehost/users/Tom` to delete all data for user Tom
- Finally, perform the GET request a third time to check that all data related to user Tom has been deleted successfully

But how can we actually perform these checks? For that, we need some sort of tool that sends the required requests to the appropriate host. This tool can be as simple and as common as a standard web browser, such as Chrome or Firefox. For example, when we open `http://api.zippopotam.us/us/90210` in a browser, we see the following result:



```
{
    post code: "90210",
    country: "United States",
    country abbreviation: "US",
  - places: [
      - {
            place name: "Beverly Hills",
            longitude: "-118.4065",
            state: "California",
            state abbreviation: "CA",
            latitude: "34.0901"
        }
    ]
}
```

This is the JSON response to a GET request sent to the `zippopotam.us` web service (a public API that returns location data based on various international zip codes).

This shows that, technically, you could use your browser as a test tool for RESTful web services. However, there are a number of downsides to this approach, the most important being:

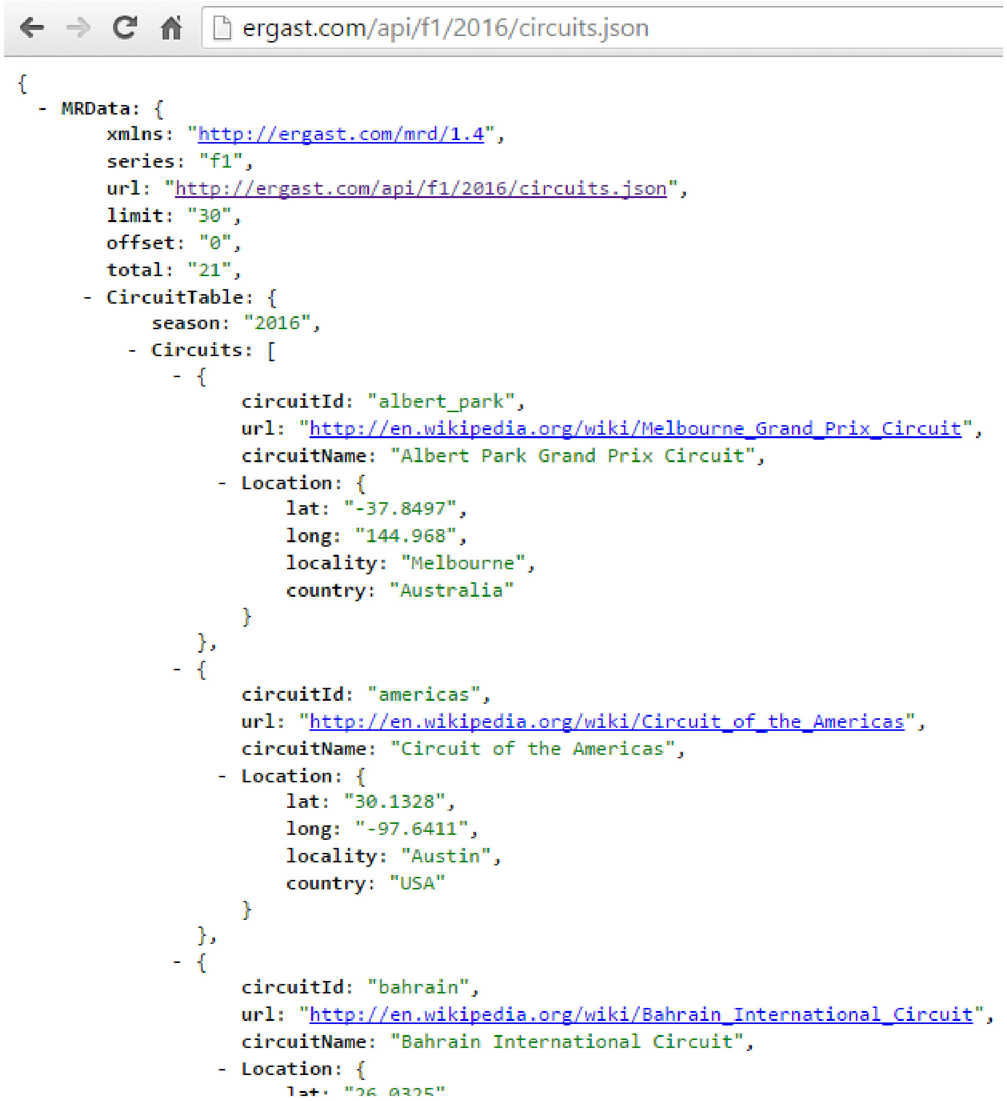
- It is quite hard to perform requests other than GET requests through a web browser (at least without additional plugins)
- Specifying request parameters, for example for POST and PUT requests, isn't straightforward
- You have to type every request (consisting of the URI and any parameters and/or payload that is sent to the web service provider) completely by hand, which makes data driven testing cumbersome
- You can't perform validations on the response data in an automated manner
- Fortunately, there are a number of suitable tools available that are far more suitable, ranging from:
- Browser plugins such as Postman for Chrome, via
- Free and open source tools such as SoapUI and REST Assured, to
- Enterprise grade tools supplied by companies such as Parasoft, HP and IBM

In the remainder of this article, we will take a closer look at REST Assured and the way you can use it to write readable and maintainable tests for a RESTful web service. We will do so by example of an API that returns historical data for Formula 1 races. Documentation for this API is available at http://ergast.com/mrd/.

Using REST Assured for automating checks on RESTful web services

REST Assured is a Java library that takes away much of the work needed to send HTTP requests to RESTful web services and capturing and validating the response, allowing you to write readable and maintainable tests. Extensive user documentation for REST Assured can be found on the tool website at `http://rest-assured.io`.

Let's see how this works by means of an example. Performing a GET request using the URI `http://ergast.com/api/f1/2016/circuits.json` will return a list of all circuits for the 2016 Formula 1 season in JSON format:


ergast.com/api/f1/2016/circuits.json

```
{
  - MRData: {
        xmlns: "http://ergast.com/mrd/1.4",
        series: "f1",
        url: "http://ergast.com/api/f1/2016/circuits.json",
        limit: "30",
        offset: "0",
        total: "21",
      - CircuitTable: {
            season: "2016",
          - Circuits: [
              - {
                    circuitId: "albert_park",
                    url: "http://en.wikipedia.org/wiki/Melbourne_Grand_Prix_Circuit",
                    circuitName: "Albert Park Grand Prix Circuit",
                  - Location: {
                        lat: "-37.8497",
                        long: "144.968",
                        locality: "Melbourne",
                        country: "Australia"
                    }
                },
              - {
                    circuitId: "americas",
                    url: "http://en.wikipedia.org/wiki/Circuit_of_the_Americas",
                    circuitName: "Circuit of the Americas",
                  - Location: {
                        lat: "30.1328",
                        long: "-97.6411",
                        locality: "Austin",
                        country: "USA"
                    }
                },
              - {
                    circuitId: "bahrain",
                    url: "http://en.wikipedia.org/wiki/Bahrain_International_Circuit",
                    circuitName: "Bahrain International Circuit",
                  - Location: {
                        lat: "26.0325"
```

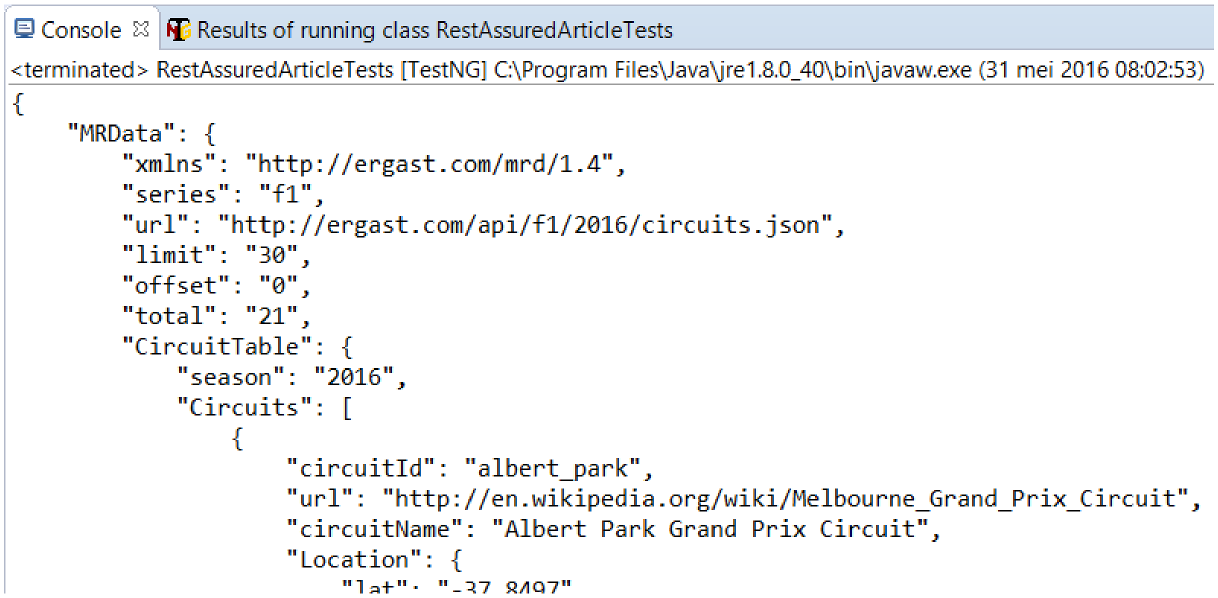
Performing the same request with REST Assured (and logging the result to the standard output) is done using the following piece code:

```
@Test
public void get2016CircuitList() {

    given().
    when().
        get("http://ergast.com/api/f1/2016/circuits.json").
    then().
        log().
        body();
}
```

Note the use of the `given()` / `when()` / `then()` syntax. This is a REST Assured feature that allows your tests to be even better readable, as well as discuss specifications and tests with other stakeholders in true Behaviour Driven Development (BDD) fashion. REST Assured hides a lot of complexity of setting up a connection to the web service provider and capturing and parsing the response returned, making the tasks of writing and reading tests very straightforward.

When we execute this test, the response that is written to the standard output is the same as the one we saw in our browser:



```
<terminated> RestAssuredArticleTests [TestNG] C:\Program Files\Java\jre1.8.0_40\bin\javaw.exe (31 mei 2016 08:02:53)
{
    "MRData": {
        "xmlns": "http://ergast.com/mrd/1.4",
        "series": "f1",
        "url": "http://ergast.com/api/f1/2016/circuits.json",
        "limit": "30",
        "offset": "0",
        "total": "21",
        "CircuitTable": {
            "season": "2016",
            "Circuits": [
                {
                    "circuitId": "albert_park",
                    "url": "http://en.wikipedia.org/wiki/Melbourne_Grand_Prix_Circuit",
                    "circuitName": "Albert Park Grand Prix Circuit",
                    "Location": {
                        "lat": "-37.8497"
```

Of course, we would also like to perform validations on the response data returned. For example, we would like to assert that Sepang is in the list of circuits returned. This can be done with REST Assured using Hamcrest Matchers, which not only provide you with a lot of different types of assertions, but also keeps your code readable:

```
@Test
public void checkSepangIsInTheListOf2016CircuitsJSON() {

    given().
    when().
        get("http://ergast.com/api/f1/2016/circuits.json").
    then().
        assertThat().
        body("MRData.CircuitTable.Circuits.circuitName", hasItem("Sepang International Circuit"));
}
```

This assertion first retrieves the collection of all elements from the JSON response that correspond to the JSON path `MRData.CircuitTable.Circuits.circuitName`, then checks whether the entry 'Sepang International Circuit' is a member of this collection.

There are lots of different Hamcrest Matchers that can be used with REST Assured, so you're bound to find one that suits your validation needs. And if you really need a different Matcher, there's always the option to create one by extending the default set of Hamcrest Matchers. By the way, these Matchers work just as well when your RESTful web service returns data in XML format:

```
@Test
public void checkSepangIsInTheListOf2016CircuitsXML() {

    given().
    when().
        get("http://ergast.com/api/f1/2016/circuits.xml").
    then().
        assertThat().
        body("MRData.CircuitTable.Circuit.CircuitName", hasItem("Sepang International Circuit"));
}
```

Some more useful REST Assured features

Now that we have covered basic response content validations, let's have a look at some other useful features offered by REST Assured for writing tests for RESTful web services.

Validating HTTP response headers

REST Assured can also validate various fields in the HTTP response header, such as the HTTP status code and the response MIME type.

Authentication mechanisms

REST Assured supports a number of authentication mechanisms, such as Basic and OAuth (version 1 and 2). This is very useful for writing test cases for secure RESTful web services.

Measuring response times

REST Assured can also measure response times for individual requests. This does not make it a substitute for proper performance testing, but high response time values for individual calls might be an early indicator for performance issues.

Further reading

For documentation on all REST Assured features you can visit the tool website at `http://rest-assured.io`. I have also written multiple blog posts featuring REST Assured on my personal blog. These posts can be found at `http://www.ontestautomation.com/tag/rest-assured/`.

Bas is a freelance test consultant with around 10 years of experience in the design and development of automated testing solutions. He is particularly interested in integration testing and testing of distributed systems and APIs. Next to his day-to-day helping out his clients Bas regularly delivers talks and workshops on a range of topics related to software testing and test automation.

He blogs at http://www.ontestautomation.com and goes by @_basdijkstra on Twitter.

Tune in now
TECHNO
TALKS
WITH LALIT
BRAND NEW SHOW: Techno Talks with Lalit on www.tvfortesters.com

Episode 1 – coming soon

Talking Mobile App Testing

with Daniel Knott

TV for Testers

Your one stop shop for all software testing videos

Sharing is caring! Don't be selfish ☺

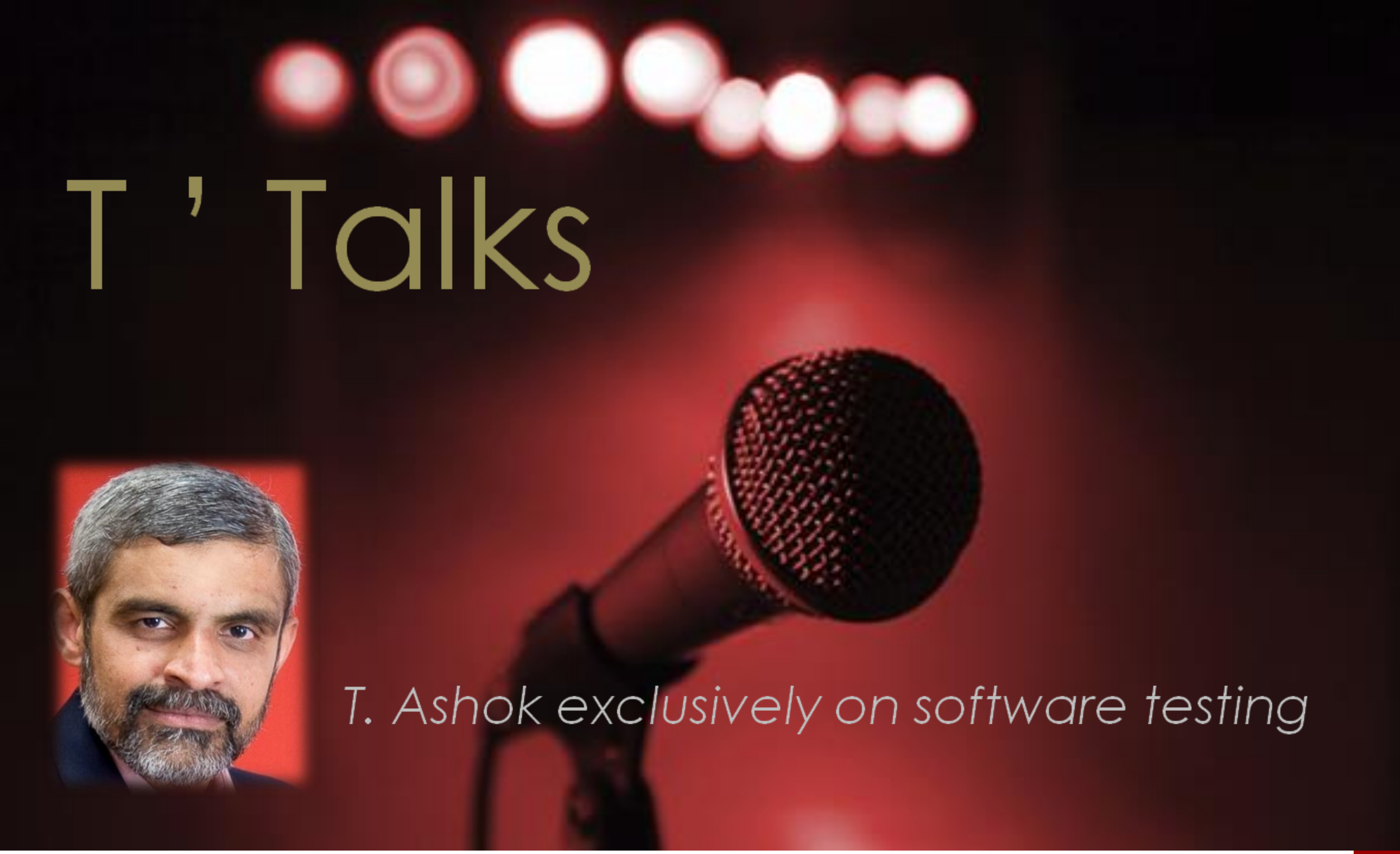
Share this issue with your friends and colleagues!

Happiness is….

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

Like our FACEBOOK page for more of such happiness

https://www.facebook.com/TtimewidTesters



Clarity of thought - Language matters

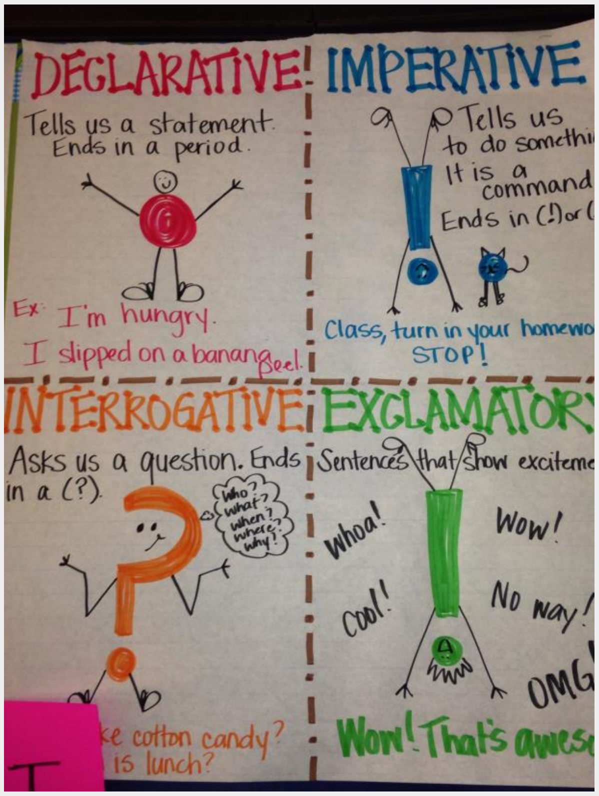
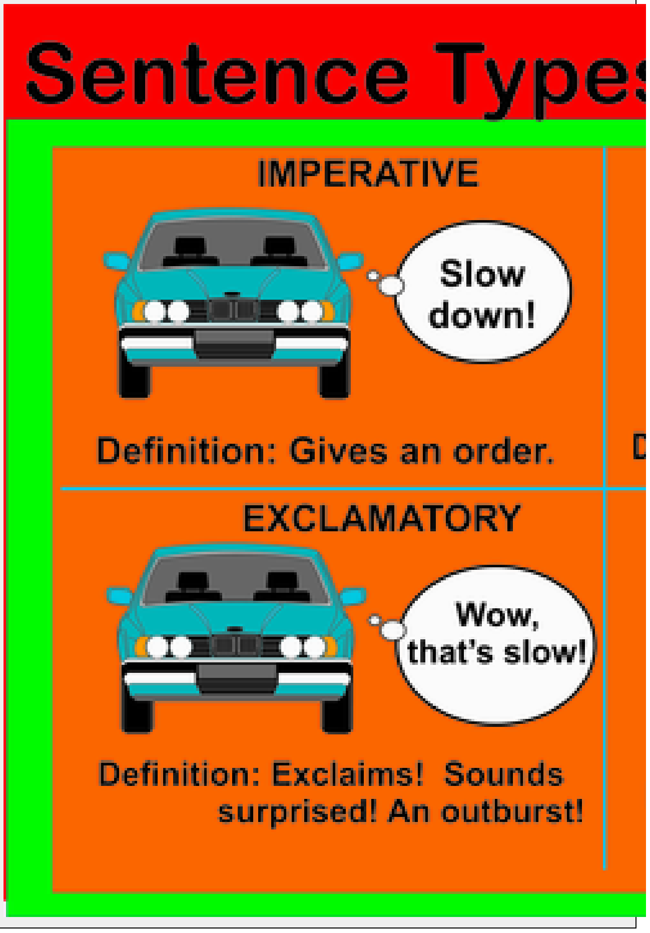
Good testing demands clarity of thought. It is about seeing the system in your mind and run through situations that may be interesting. It is about seeing the gaps, the missing elements that results in questions to aid in completing the picture. It is about transforming your mind to that of the end user(s) and analysing the system.

Good clarity is like water, clear and transparent, dirt can colour it but not alter it. It flows freely taking any shape without losing itself. To be like water is Zen-like, a state when the creative juices flow best.

Language plays a key role in clarity, it is the vehicle that enables us to express our inner thoughts. The style of sentence used to express one's thought communicates clear intent and ensure clear transmission of information to the recipient.

So what are the sentence types?

The four sentence types are Declarative, Imperative, Interrogative and Exclamatory. The chart alongside illustrates clearly as to what each sentence type is. Note that the each sentence type has a clear purpose of its intent

Source: http://pin.it/BtA3E6Z	Source : http://bit.ly/29IHWEV
	

DECLARATIVE
Tells us a statement. Ends in a period.
Ex: I'm hungry.
I slipped on a banana peel.
IMPERATIVE
Tells us to do somethi
It is a command
Ends in (.) or (
Class, turn in your homewo
STOP!
INTERROGATIVE
Asks us a question. Ends in a (?).
Who? what? when? where? why?
ke cotton candy?
is lunch?
EXCLAMATORY
Sentences that show exciteme
Whoa!
Wow!
Cool!
No way!
OMG
Wow! That's awes

Sentence Type
IMPERATIVE
Slow down!
Definition: Gives an order.
EXCLAMATORY
Wow, that's slow!
Definition: Exclaims! Sounds surprised! An outburst!

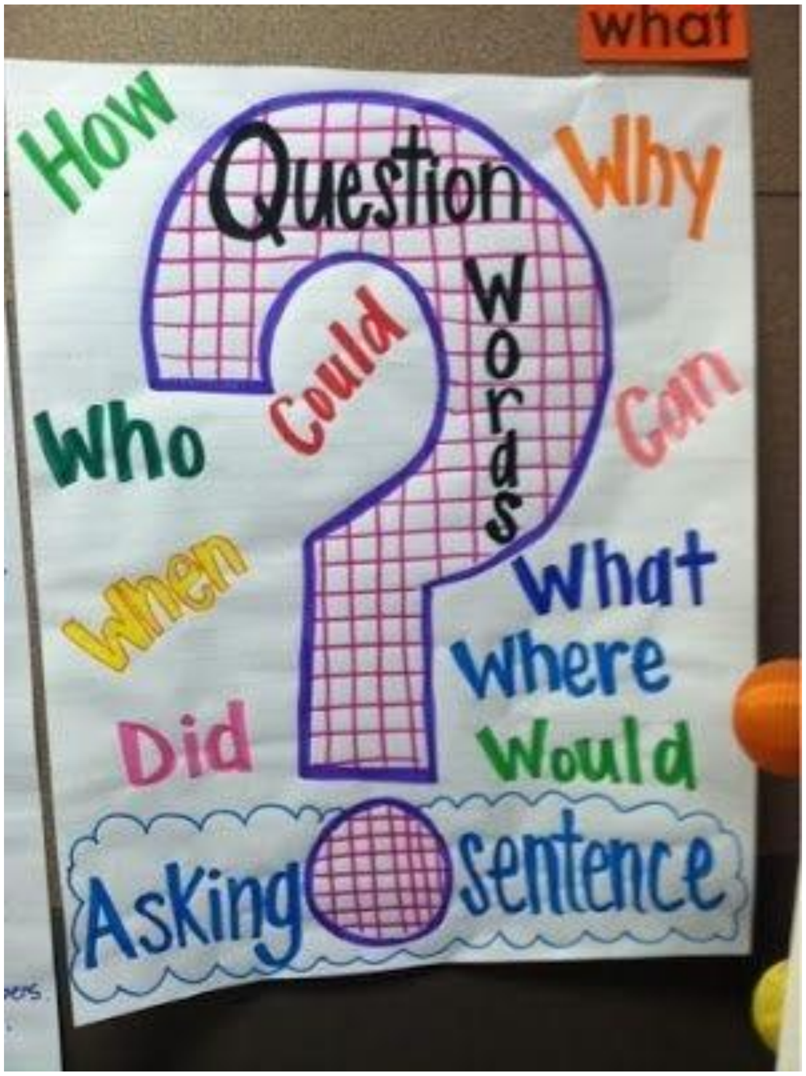
The type specifies the structure of sentence, setting up a clear intent and therefore sharpening the semantics of the content. So choosing the right sentence type in the various different stages of the test cycle ensures you stay clear by focusing on the right intent.

Let us see analyse the intent at different stages of testing and therefore choose the sentence type that best express this.

1 Understanding

At the early stage of understanding where we are analysing, reading, exploring the system, our mind is filled with questions.

Source: http://pin.it/WfPxCTg

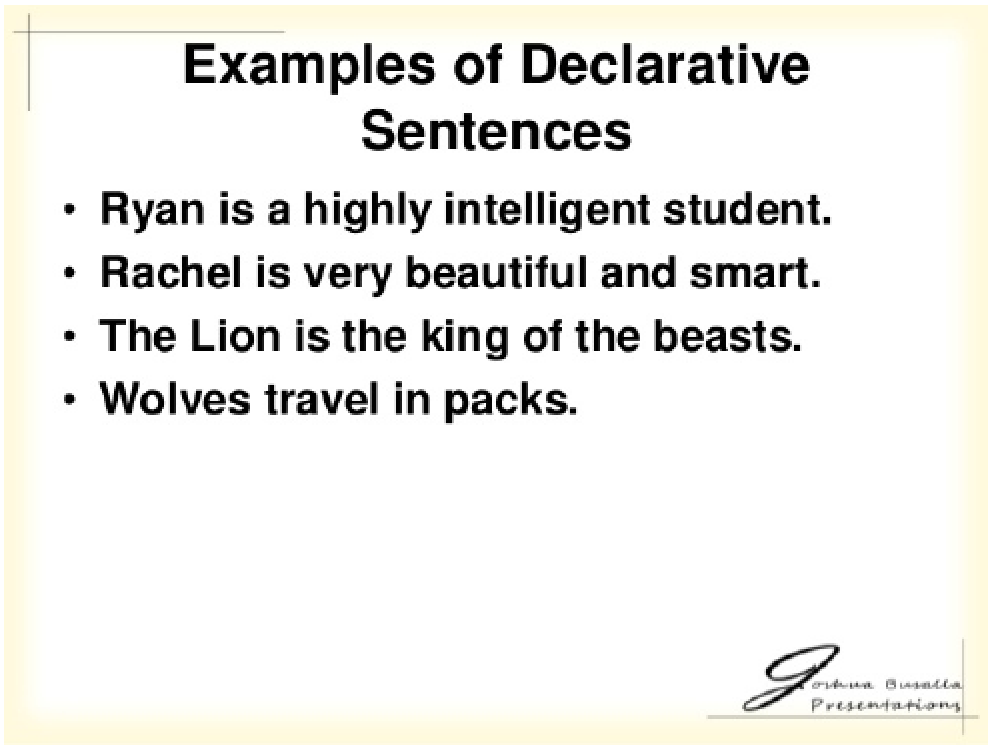


At this stage, our sentence is the “asking sentence” i.e. Interrogative.

2 Approach/Strategy/Plan

Once we understand and want to outline the approach/strategy/plan we need to describe it. The sentence type that serve this purpose well is ‘Declarative’. Note that the style of writing a typical specification is ‘ Declarative’.

Source: http://bit.ly/29IOnb2



Examples of Declarative Sentences

- Ryan is a highly intelligent student.
- Rachel is very beautiful and smart.
- The Lion is the king of the beasts.
- Wolves travel in packs.

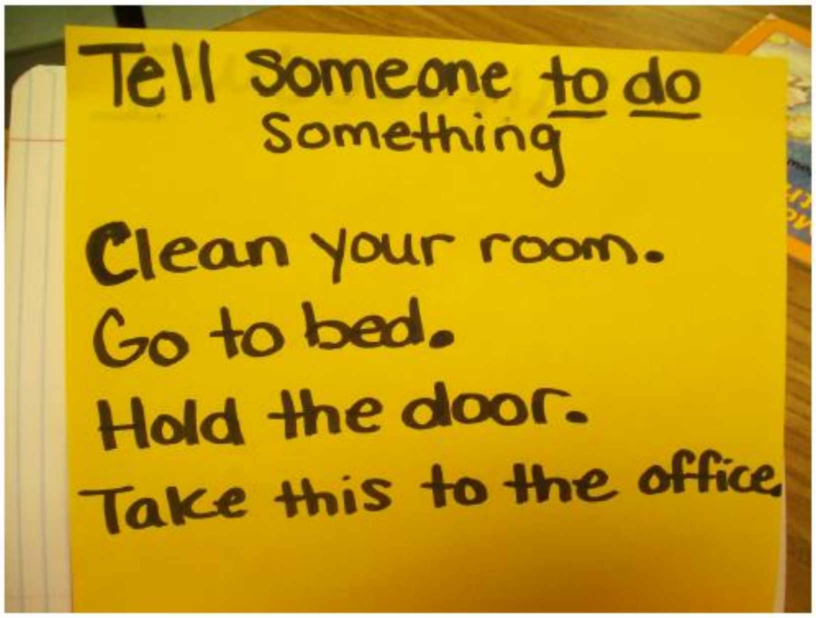
Joshua Busalla Presentations

3 Design

Once we understand, we are ready to design test scenarios/cases with the clear purpose of evaluation of the system. The evaluation scenario is best written as an action to be performed, a command really. The type of sentence that best suits this intent is 'Imperative'. e.g. "Ensure that the system does blah when bleh"

If you are going to describe the procedure of evaluation, it is best done as set of action steps. What is the best suited for this? Imperative, of course.

Source: http://bit.ly/29IOpQk



Tell Someone to do Something

Clean your room.

Go to bed.

Hold the door.

Take this to the office

4 Defect reporting

With the intent of describing as to what the defect is, the sentence that is best suited is 'Declarative'. On the other hand, describing how to reproduce the defect is about doing a set of actions, best communicated by 'Imperative' sentences. Avoid 'Exclamatory' sentences here so as to stay professional and not upset human emotions!

5 Test Reporting

As in any reporting, describing facts professionally is of essence without getting entangled into emotions. Declarative sentences are best suited for this.

6 Discussion/Meeting

This is an interesting communication channel that involves making informative statements to enable understanding (Declarative), asking questions to clarify (Interrogative), making suggestions (Imperative) and expressing emotions (Exclamatory - Be careful here!)

Writing is often a reflection of the mind, good writing implies an uncluttered mind. Our job as a tester is interesting, buffeted by strong headwinds like incomplete information, incorrect information, and we are expected to stay clear on the course. Clear choice of language especially sentence types can cut through the wind and ensure that we stay on course.

So pay attention to the choice of the sentence types, they matter.

May you be like water!

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

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

stag

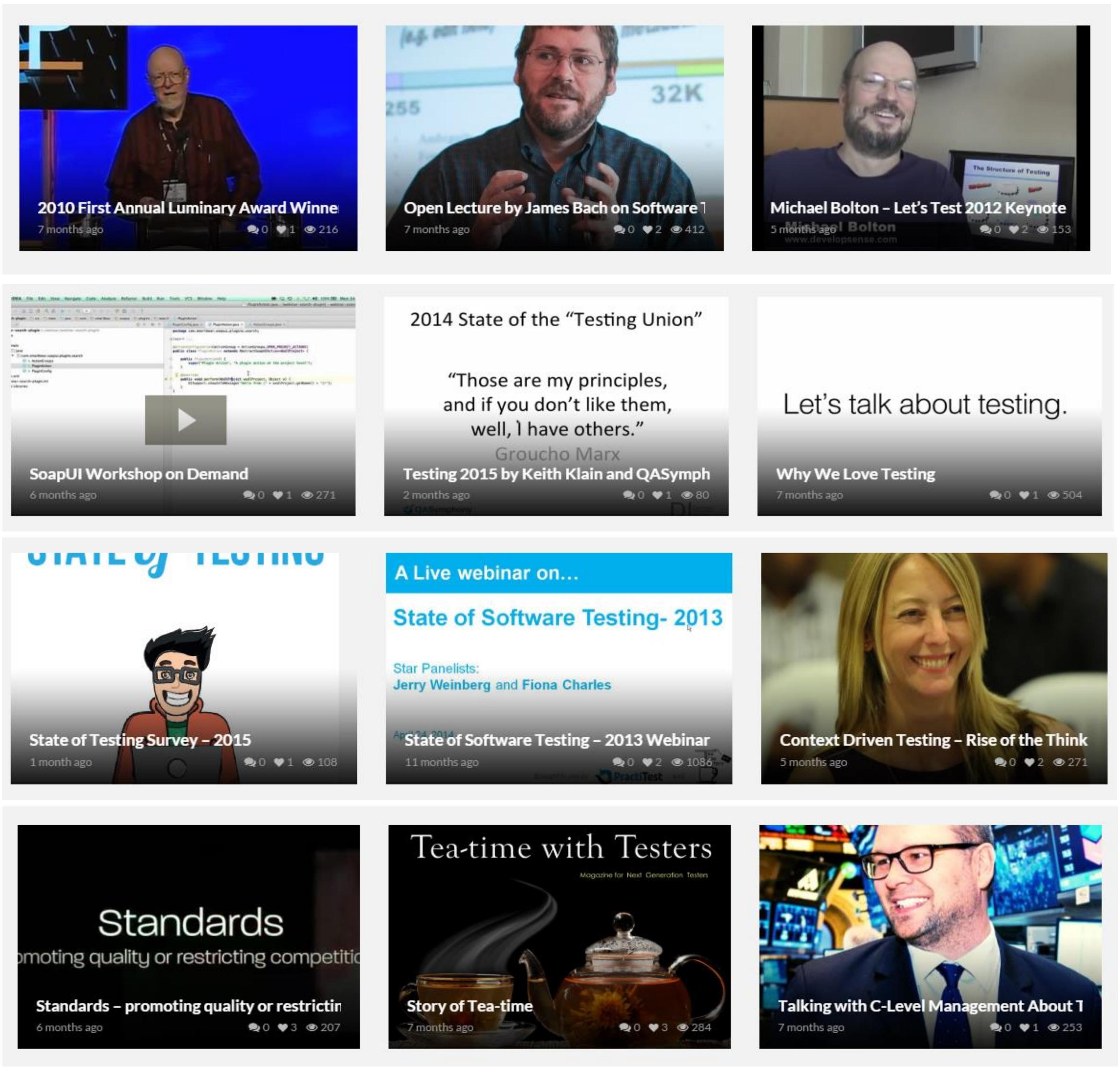
He can be reached at **ash@stagsoftware.com**

Back To Index

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

TV for Testers

Your one stop shop for all software testing videos



2010 First Annual Luminary Award Winne
7 months ago 0 1 216

Open Lecture by James Bach on Software T
7 months ago 0 2 412

Michael Bolton – Let's Test 2012 Keynote
5 months ago 0 2 153

SoapUI Workshop on Demand
6 months ago 0 1 271

Testing 2015 by Keith Klain and QASymph
2 months ago 0 1 80

Why We Love Testing
7 months ago 0 1 504

State of Testing Survey – 2015
1 month ago 0 1 108

State of Software Testing – 2013 Webinar
11 months ago 0 2 1086

Context Driven Testing – Rise of the Think
5 months ago 0 2 271

Standards – promoting quality or restrictin
6 months ago 0 3 207

Story of Tea-time
7 months ago 0 3 284

Talking with C-Level Management About T
7 months ago 0 1 253

2014 State of the "Testing Union"
"Those are my principles, and if you don't like them, well, I have others."
Groucho Marx
Let's talk about testing.
A Live webinar on…
State of Software Testing- 2013
Star Panelists:
Jerry Weinberg and Fiona Charles
Standards
promoting quality or restricting competitio
Tea-time with Testers
Magazine for Next Generation Testers

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.

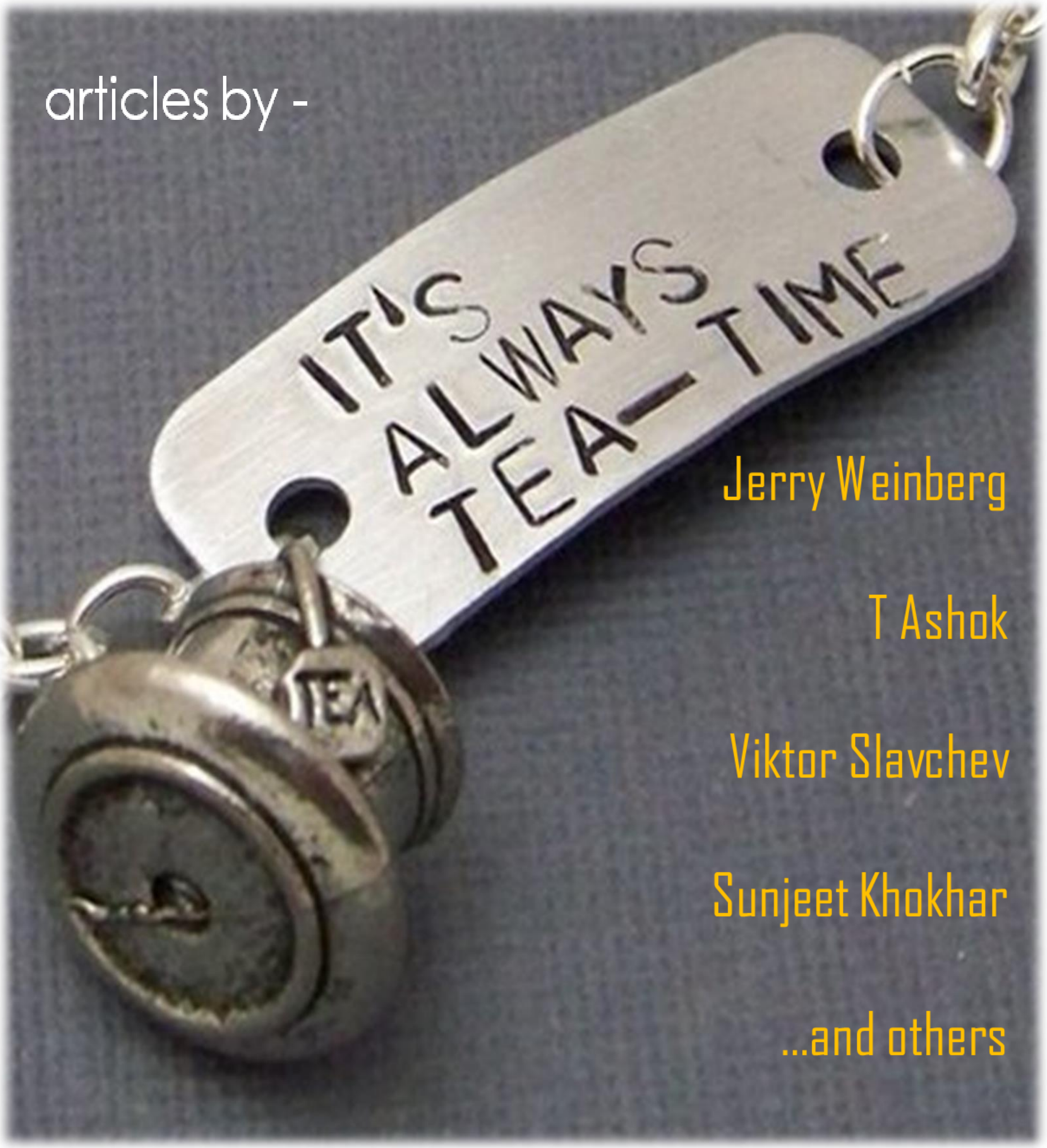
And the **Good News** is…

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 nexxt issue



articles by -

Jerry Weinberg

T Ashok

Viktor Slavchev

Sunjeet Khokhar

...and others

our family

Founder & Editor:

Lalitkumar Bhamare (Pune, India)

Pratikkumar Patel (Mumbai, India)

Lalitkumar

Pratikkumar

Contribution and Guidance:

Jerry Weinberg (U.S.A.)

T Ashok (India)

Joel Montvelisky (Israel)

Jerry

T Ashok

Joel

Editorial | Magazine Design | Logo Design | Web Design:

Lalitkumar Bhamare

Cover page image – Shutterstock

Core Team:

Dr.Meeta Prakash (Bangalore, India)

Dirk Meißner (Hamburg, Germany)

Dr. Meeta Prakash

Dirk Meißner

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

Give Feedback