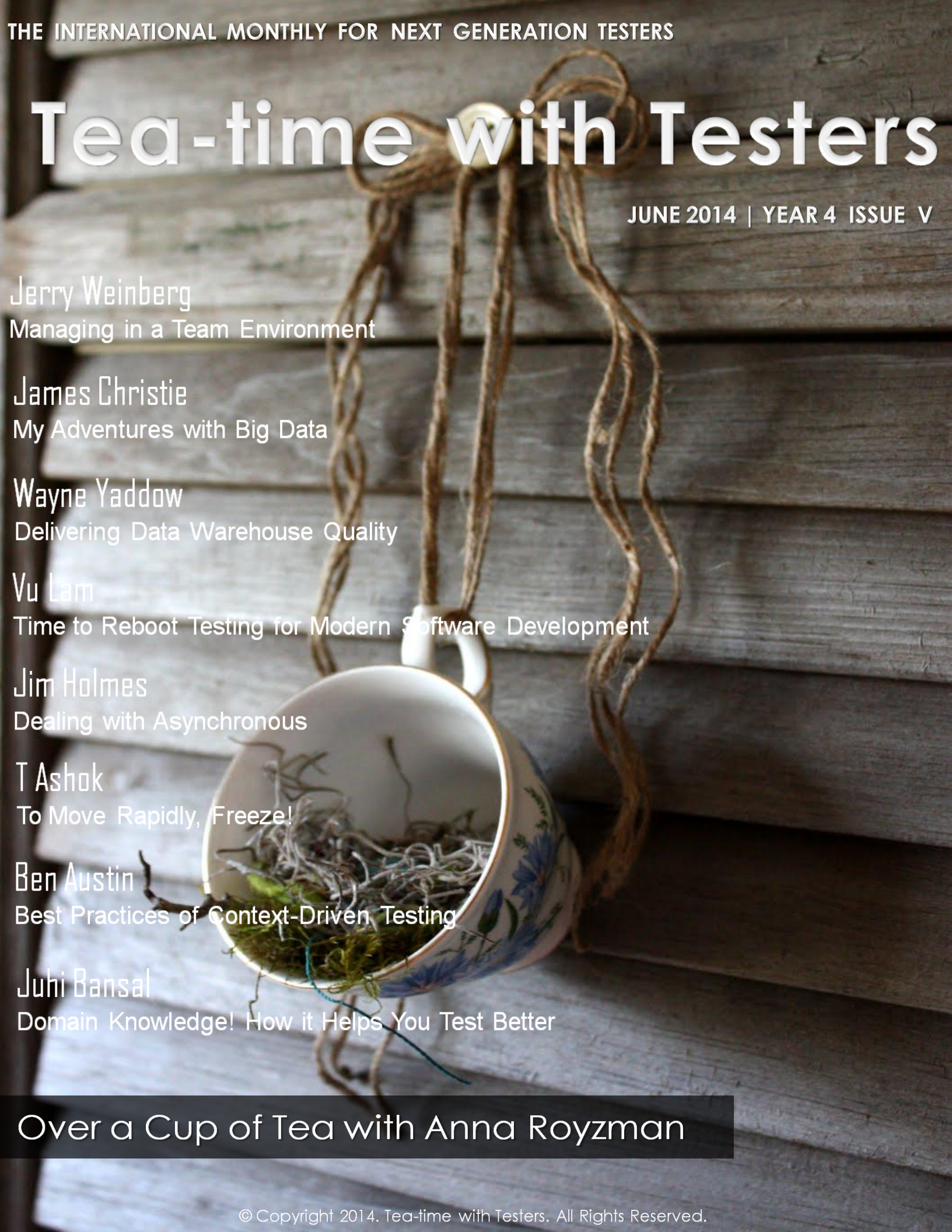
THE INTERNATIONAL MONTHLY FOR NEXT GENERATION TESTERS

Tea-time with Testers

JUNE 2014 | YEAR 4 ISSUE V



Over a Cup of Tea with Anna Royzman


Musings over Tea-time
Anthology of T-Talks
T Ashok
DOWNLOAD YOUR FREE COPY
CLICK HERE
Celebrating THREE Years of Tea Time with Testers
ANNIVERSARY SPECIAL

August 11 - 13, 2014
New York, NY, USA

CAST

Association for Software Testing

"The Art and Science of Testing"

webCAST 2014

- Keynotes
- Sessions
- "CAST Live"

August 12-13, 2014
9 am - 7 pm EDT (GMT -4)

www.AssociationForSoftwareTesting.org/webCAST

TEA-TIME WITH TESTERS

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

Created and Published by:

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

Editorial and Advertising Enquiries:

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

Editorial

Let's meet to explore the Art and Science of Testing!

If you are a tester who keeps himself updated with latest in Software Testing then you must be knowing that **Conference of Association for Software Testing (CAST)** is happening at New York this August.

I have special respect and liking for CAST because of its nature, uniqueness and because every year it's organized by people whom I admire and respect the most. I regret for missing an opportunity to speak at this most anticipated conference last year but I am glad that I am going to attend it this time.

CAST 2014 is special for me as I am going to meet awesome testers and testing gurus whom I have known, seen and have interacted with only over internet. Needless to mention the opportunities that I would get there i.e. to confer, to learn from peers and to socialize.

I wish I could invite you to attend this conference but unfortunately it is sold out. But hey, you can still attend it and that too at no cost. Yes, it will be LIVE and it's gona be absolutely FREE!

Through Tea-time with Testers; you have been reading articles from James Bach, James Christie, Fiona Charles, Keith Klain, Anna Royzman (just to name a few). Wouldn't it be amazing to see them talking live about testing?

I strongly recommend you to take benefit of live streaming from CAST. All you need to do is **subscribe for webcast** or you can keep an eye on **this** space…

That's all from my end for now. I promise to bring some interesting testing stories for you; as I come back from CAST. Don't forget to say me hello if you happen to be there. ☺

Yours Sincerely,

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

QuickLook

Editorial

What's making News?

Tea & Testing with Jerry Weinberg

Speaking Tester's Mind

In the School of Testing

T ' Talks

Over a Cup of Tea with Anna Royzman

Family de Tea-time with Testers

Windows bug-testing software cracks stem cell programs

- by **Paul Marks**

Newscientist: SOFTWARE used to keep bugs out of Microsoft Windows programs has begun shedding light on one of the big questions in modern science: how stem cells decide what type of tissue to become.

Not only do the results reveal that cellular decision-making is nowhere near as complicated as expected, they also raise hopes that the software could become a key tool in regenerative medicine.

"It is a sign of the convergence between carbon and silicon-based life," says Chris Mason, a regenerative medicine specialist at University College London. "World-class stem cell scientists and a world-class computer company have found common ground. It is work at such interfaces that brings the big breakthroughs."

Stem cells are the putty from which all tissues of the body are made. That means they have the potential to repair damaged tissue and even grow into new organs.

Embryonic stem cells hold particular promise as they can either renew themselves indefinitely or differentiate into any kind of cell in the body – a property known as pluripotency.

The process that sets a stem cell on the path to either self-renewal or differentiation was thought to be a highly complex web of genetic and environmental interactions. That web is known as the interactome.

Embryonic stem cells are currently being trialed as a way to restore vision and treat spinal injury. But these trials, and others in the pipeline, are hampered by the fact that no one really knows what determines the fate of any particular stem cell. Today's techniques for making a stem cell differentiate into a certain tissues are hit-and-miss, says Mason.

What's needed is a more deterministic, reliable method, says Sara-Jane Dunn, a computational biologist at Microsoft Research in Cambridge. One approach is to frame the problem in the language of computation. The genetic and environmental cues that determine the cell's fate can be thought of as inputs, with the cell itself as the processor, Dunn says.

Stem cells' capacity to renew themselves is the simplest of the two possible paths out of the pluripotent state. To find the program behind this, Dunn, along with stem cell scientists Graziano Martello at the University of Padua in Italy, and Austin Smith at the University of Cambridge, tried to isolate the genetic and environmental processes at work in mouse embryonic stem cells.

They used a technique pioneered at Smith's lab that uses cultures of various inhibitory proteins to keep embryonic stem cells continually renewing themselves rather than differentiating into other cells. The team immersed the stem cells in four different types of these cultures and analysed which genes they expressed in which environment, and to what extent.

Next, to uncover the program that kept the cells in the unspecialised state, they turned to a mathematical technique called formal verification. Originally developed to detect and remove errors in software that keeps aircraft aloft and nuclear power plants safe, the technique is now widely used to eliminate bugs in commercial software, such as Microsoft's Windows packages.

Formal verification examines the algorithms in a piece of software to check that the output will always be what the programmer intended. But it can also work back from the output to infer the nature of the algorithm creating it – just what Dunn's team required.

The team rewrote the Microsoft formal verification program, then fed genetic and chemical data from the different stem cell cultures into it – with some surprising results. There appears to be no highly complicated interactome behind self-renewal. Instead, the stem cells' program involved just 16 interactions between 12 proteins, called transcription factors, and three environmental inputs, in this case provided by chemicals in the lab. The relative simplicity of the process means biologists have a much greater chance of reliably influencing stem cell fate.

The researchers also found they could use the software – called the Reasoning Engine for Interaction Networks (RE:IN) – to predict with about 70 per cent accuracy how the cells would respond to genetic changes. For instance, they were able to predict whether the cell would remain pluripotent after knocking out one gene, or two (*Science*, doi.org/s5f).

The next step is to work out the underlying biological processes behind stem cell differentiation. Smith and Martello plan to encourage mouse embryonic stem cells to turn into neurons, using Microsoft's formal analysis tools as a guide. They will also use it to study cell reprogramming, in which an adult cell is converted back into its pluripotent state.

"It's remarkable. I have never seen anything like it," says Mason. If this technique reveals the molecular program behind differentiation, it might enable us to do it in the lab more robustly, he says.

Simon Tomlinson from the Institute for Stem Cell Research in Edinburgh, UK, agrees that this is a big step forward. "It signals a future where many discoveries in this area are driven by predictive model building."

The real test will be in repeating the results with human embryonic stem cells, says Robert Lanza at Advanced Cell Technology in Marlborough, Massachusetts, who is involved with the vision trial. "It's usefulness will have to be determined over time."

This article appeared in print under the headline "Anti-bug software foretells cell fate"

Back To Index

A million dollar smile ?

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

*Adverts starting from $100 USD | *Conditions Apply

Photo credits: Amir Eltahan

I got to know about **Anna Royzman** when I joined Association for Software Testing (AST) and learned about their Special Interest Group (SIG). In 2012, Anna started AST Leadership SIG, and serves as the SIG Chair.

From what I know about Anna, there seems to be almost no activity in testing field which she does not have experience of. Anna organizes discussion panels, leads SIGs, creates workshops, and speaks at conferences to promote the value of skillful testing and the whole team approach to quality. She made her speaking debut at Emerging Topics at CAST 2011.

You must have read Anna's article in Tea-time with Testers' **Women in Testing** special edition. That was the first time I got to interact with Anna and later I got to work closely with her in AST's BBST Instructors course.

I feel privileged for getting yet another opportunity to work with Anna. She is Conference Co-Chair of **CAST (Conference of Association for Software Testing**) this year. While discussing about CAST; we also discussed testing and this interview is an outcome of our dialogs.

Read on to know more about Anna, what she feels about different things in testing and about CAST 2014.

- Lalitkumar Bhamare

Over a Cup of Tea with Anna Royzman

You are known as one of the influential women and leaders in testing community. We are curious to know about your journey in software field.

Thank you, Tea Time! My journey as a leader started before the journey in software field. I came to US in 1997, went to college to study Recreational Leadership, and worked as the Summer Camp Director for few years. This practice taught me a lot about being a people manager and a leader. Later on, I started my career in software field, and was fascinated by testing -- to the point that I got the job as a software tester. It was a matter of time before I became a test manager.

It took me many more years to become a context-driven test strategist. It's what I enjoy doing, and it better defines my current job responsibilities.

Would you like us to tell more about your involvement with (AST) Association for Software Testing?

AST is a volunteer organization, and all the work we do is dedicated to the advancement of our testing community.

In 2012, I started AST Leadership SIG (special interest group) and serve as the SIG chair. This year, our SIG is working on a project called "Talking to Management about Testing". We interview industry experts and share their stories and advice with the testing community.

Also, I co-chair the 9th Annual Conference of the Association for Software Testing (CAST). Chairing CAST is a special opportunity, which may come once a lifetime, since this conference is organized in different city each year. This year it's in New York City, August 11th-13th.

Looking after such responsible positions along with one's routine job at workplace makes things tough at times. How do you manage these multiple responsibilities and what motivates you?

I would never call my job a routine – otherwise I would not stay there for long. Regarding the community engagement; there comes a time, when you feel the need to give back to the community and grow your future. That feeling motivates me. I bring the experience from engaging in community events back to work, and make it a better place for testers as well.

Are there any interesting stories that you would like to share with our readers, especially our women readers?

Few months ago I attended an event with the User Experience designers. It was an "open space" type of conference, where the participants create the content through discussing topics which are of most interest for them. One of the sessions was on how UX designers can better understand software developers, and vice versa. What stroked me was how closely their discussion reflected the issues, which we, the testers, discover in working with developers as well. Most of the confusion and misunderstanding came from the fact, that some designers, as well as testers, don't write code. The participants found the solution in defining how the designers approach their tasks, what drives their thinking, and how they construct their designs. Since there were some developers in the audience, they did the same – and the whole group agreed that it was the point of contact. I learned a thing or two from this discussion: explaining to a developer how you arrive at a testing strategy is a more constructive conversation than that of 'my job is finding bugs in your code'. Take the emotions out, highlight the skills. We are all going to win this way.

You had written an excellent article around 'Peer Pressure' for Women in Testing special edition of TTwT. Can you tell us in short how important do you find peer pressure in one's career development? Any names of peers that you would like to mention?

.... CONTINUED ON PAGE 50

Tea & Testing with Jerry Weinberg

Managing in a Team Environment

A team effort is a lot of people doing what I say. - Michael Winner, British film director

Many managers seem to think they are Hollywood directors and share the view that team effort means everyone will be subservient to their high-and-mightiness. At one company, the managers showed their team spirit by having parking places closer to the door than the parking spaces for people in wheel chairs! For managers with such attitudes, this article will not help.

Other managers, however, realize that teamwork means more than following orders from the Emperor, yet they unconsciously undermine their own efforts to reap the benefits of effective teams. This article points out some of those unconscious behaviors and suggests some conscious behaviors to enhance team performance.

1. The Manager's Role in a Team-Based Organization

Perhaps the most common confusion about the manager's role in a team-based organization is between the manager and the team leader. The team leader (or, if a self-managed team, the entire team) is responsible for the technical task of the team. The manager, on the other hand, is responsible for nontechnical direction of two or more teams. Seen from inside the team, the manager's role is to unburden the team leader by handling certain nontechnical tasks. Seen from the manager on the outside, the manager's role is to align the team with the higher goals of the organization.

1.1 Delegation

The manager initiates work units by delegating standard task units to teams. These standard task units are specified in terms of the prerequisites that must be in place for the task to be completed, such as requirements, human resources, tools, products of earlier tasks, working space, funds, and training.

1.2 Control

Each task unit must also specify the process (usually a review of some kind) by which the product of the task will be measured. The manager's job is to work with the team to establish control points based on these measurements and to monitor these check points externally. The manager steps inside the team only when some checkpoint is not reached, or when a team breakdown occurs.

1.3 Coordination with the rest of the organization

The manager is the primary coordinator of issues between teams, which doesn't mean that the manager must directly supervise all communication between teams. The manager's most important functions are to ensure commitments between teams consider the larger context, decision processes are appropriate, and decisions are explicitly documented.

Coordinating between teams is part of a more general role of acting as the team's buffer to the outside world. As part of this role, the manager handles company policies, budgets, and personnel administration.

2. Delegating Work

According to Katzenbach and Smith, the number one way to build teams is to delegate challenging work: Teams do not become teams just because we call them teams or send them to team-building workshops.

In fact, many frustrations with broad-gauged movements toward team-based organizations spring from just such imbalances. Real teams form best when management makes clear performance demands.

Challenge is an emotional reaction, influenced as much by the way the task is assigned as by the task itself. And earlier volume in this series laid out the physical structure of a standard task unit, but gives no guidance on the question of the emotional structure. For success with teams of mixed temperaments, tasks must be delegated in a way that is challenging, clear, and supportive.

2.1 Challenging

The SP Troubleshooters are the ones most challenged by the task itself, while the NT Visionaries are most concerned with communicating the challenge to the rest of the team. Here are some of the things NTs and SPs want from their managers in order to be challenged:

•Don't ever give us an insultingly easy task. It must be difficult, but not impossibly difficult and especially not made extra difficult by rigid constraints that have nothing to do with the task itself.

•Give us tasks defined in terms of results, not the methods of achieving those results. This definition should be a vision of the problem to be solved, a definition that they can interpret in a way that is meaningful to them, and will satisfy them when they achieve it.

•We'll accept external control checks. We especially welcome quality checks and will tolerate time checks. We do not appreciate budget checks unless made part of the challenge.

•We don't mind complication, but not complication created by changing the rules in the middle of the game— unless those rule changes can be related to the vision of the problem to be solved. Challenge us with unfamiliarity is a challenge, but make allowance for learning.

•Allow us to be creative and enjoy ourselves.

2.2 Clear

For the SJ Organizers, challenge is all good and well, but it is essential that the challenge be expressed clearly, right from the beginning. They may express what they need from their managers in the following way:

•Be up front, giving precise and clear instructions that are complete in all details.

•Put everything in writing. Explore all possibilities, but simplify what you present to us.

•Be available to answer questions, because our first two needs are never really met.

2.3 Supportive

The NF Team builders want challenge and clarity, too—but mostly because the others want them. They are less interested in the task than in the environment in which the task is to be done. Here's what they would desire from the manager who delegates a task to them:

•The most important thing you can do is hope for our success and structure the task so that success is possible, perhaps in small increments as the task proceeds.

•Balance the workload, so that our team has a fair share with others. Give us something that fits in a balanced way with the skills we have on our team. Don't ask us to do things we are unable to do, but especially don't ask us to do things we aren't willing to do.

•Trust the team to do the job, but deliver the resources you promise.

•Give feedback, whatever flavor, but be generous in interpreting what you see.

•Protect the team from outside demands, and provide guidance in steering through the organizational waters.

2.4 Mistakes in delegating

Many new managers have told me that the thing that surprised them the most in their new role was the amount of work it takes to get someone else to do work. It's hard enough to satisfy the diverse demands of the different temperaments, but as a manager you also have to behave in ways that satisfy some universal human needs:

•No matter how clear you've tried to be, you will be misunderstood. You must be available to answer questions, never losing patience, no matter how dumb the questions seem to be.

•You have to be prepared to enter into other people's favored communication modalities, as when they draw you a picture of what they think you said, or say in words what you drew for them in a picture.

•No matter how much work you've done, you'll make some mistakes. Admit your own your mistakes and accept their corrections.

•You must listen to complaints without being defensive or taking them personally. People will be frustrated, overworked, befuddled about what you want, anxious about meeting the schedule, protective of each other, and angry because you're "not listening" to them. When they've finished complaining, you'll have to nudge them into problem-solving mode, and be prepared to make compromises.

If you do all these things, you may begin enjoying modest success in delegating to others and watching them actually doing the task you assigned. But you won't be able to rest long, because as soon as they start working, they may slip off track, and you may be called upon to do a little steering.

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 **Software Test Professionals first annual Luminary Award.**

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

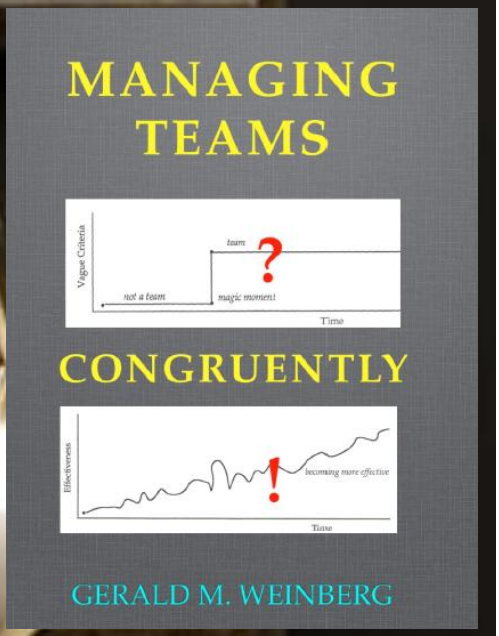
Gerald can be reached at hardpretzel@earthlink.net or on twitter @JerryWeinberg

To be effective, team managers must act congruently. These managers must not only understand the concepts of good software engineering and effective teamwork, but also translate them into their own practices. Effective managers need to know what to do, say what they will do, and act accordingly. Their thoughts and feelings need to match their words and behaviors.

And how should they do that? Jerry has shared this secret in his **MANAGING TEAMS CONGRUENTLY** book.

Its sample can be read online here.

To know more about Jerry's writing on software please click here .



TTWT Rating: ★★★★★

TEA-TIME WITH SMARTBEAR

About this column…

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

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

Best Practices of Context-Driven Testing

- by **Ben Austin**

First of all, before anyone's head explodes, let me explain what I mean by "Best Practices." I know that, particularly in the realm of context-driven testing, this term is looked upon as a major misnomer that is only spewed by the ignorant and uneducated. But I don't think that has to be the case.

Not everything should be quantitative. Not everything has to be proven without a shadow of a doubt. Most of all, we shouldn't be afraid of being wrong. For years people have subscribed to this idea that "there are no best practices," and have genuinely feared the shunning that would come with stepping outside the lines of that very narrow, absolutist mindset. Particularly in an industry so tightly tied to *quality* and *context*, it seems bizarre that no one is willing to allow any gray area when it comes to a slightly subjective term like "best practices."

For the sake of this blog post, let me briefly explain what I mean by "best practices." I'll do so by paraphrasing Cyrus Shepard, who I think has a good explanation for how we should look at this term, and what we should expect from the guidelines that fall under it:

- Best practices are a set of rules or guidelines that have consistently shown superior results for a practitioner. This doesn't mean that best practices are the only way you could or should accomplish a task, just that they have generally shown consistently higher results than other techniques.
- Best practices can help to serve as benchmarks for the industry they're applied to. They should be seen as goals for practitioners to strive for in order to raise the standard of work and enable an industry to mature over time.
- Best practices are temporary. Saying that something is a best practice today doesn't mean that I'm forever bound to that methodology. Rather, best practices should be expected to evolve over time. What is a best practice today likely won't be a best practice a decade from now. And that's okay. That means practitioners have improved with time and have created a new standard for the expected quality of their work.

And finally, best practices can, and sometimes should be, be ignored. As is the fundamental element of context-driven testing, you have to decide whether or not they fit into each project you work on. Sometimes you don't have the time or resources to do all of them. But just because they can't be blindly applied to every situation doesn't mean they don't exist. They do. And here is my take on five of the best practices of context-driven testing.

Ask Questions

This has to be the one thing that I hear context-driven testers emphasize more than anything else. Ask questions of stakeholders. Ask questions of the development team. Ask questions of your fellow testers. Without asking a slew of questions it's extremely difficult to understand the context of a project, which in turn makes it very difficult to obtain maximum test coverage.

Asking questions can also be a major driver for improving your career beyond a specific project. Constantly asking questions and questioning the status quo allows junior testers to learn from their mentors, and it allows mentors to learn from junior testers. As Keith Klain explained to me during an interview in March, the most successful testers are generally the ones who have an unquenchable curiosity and are able to spread the knowledge they've gained over the years to those around them:

The people I've seen successful in the [tester mentoring] role are folks who are conduits to knowledge. So it's like, "Let's learn from my experience, and ask a lot of questions." I think the Socratic approach to mentoring is really important to get people to learn things on their own and help them be very self-reflective and figure out what you are contributing to this problem and how you can help them tease out their own solutions. Because that's ultimately what you're trying to do anyway.

Plan Ahead

What good is all that knowledge if you don't have an efficient way to put it to use? Creating and sharing your test plan with the rest of your team and project stakeholders not only makes you more efficient, it builds rapport with the rest of the company and spurns more meaningful conversations.

No, you shouldn't be expected to take feedback from every junior developer or project manager in the organization, but sharing your initial test plans with the most prominent stakeholders gives them real insight into the type of return they should expect to see. It shows them that there are some guidelines to what your team is doing, and that any changes in their own plan will impact the testing strategy.

As JeanAnn Harrison explained it during her most recent visit to the SmartBear office, "If you plan out your testing strategy and figure out what kinds of tests you want to do then, really, you will go ahead and create a very efficient process."

Adjust Your Plan Accordingly

Just because you've made a plan does not mean it's set in stone. Rarely does a software project go entirely as expected, so you have to be ready to make adjustments as they come. Failing to be at least somewhat flexible with your test strategy will likely lead to all around frustration and less thorough test coverage. Schedules change, features are added, new priorities arise, and your strategy should adapt accordingly.

That said, if you've laid out your testing plan and shared it with stakeholders, you shouldn't be expected to bend over backward to make up for mistakes in the earlier stages of the project. Remember that your ultimate goal is to achieve the maximum test coverage you can achieve given the parameters at hand. If those parameters change through the course of the project, you will be doing yourself and your organization a disservice by staying true to a plan that no longer serves that ultimate goal.

Stakeholders Decide when a Project is Over

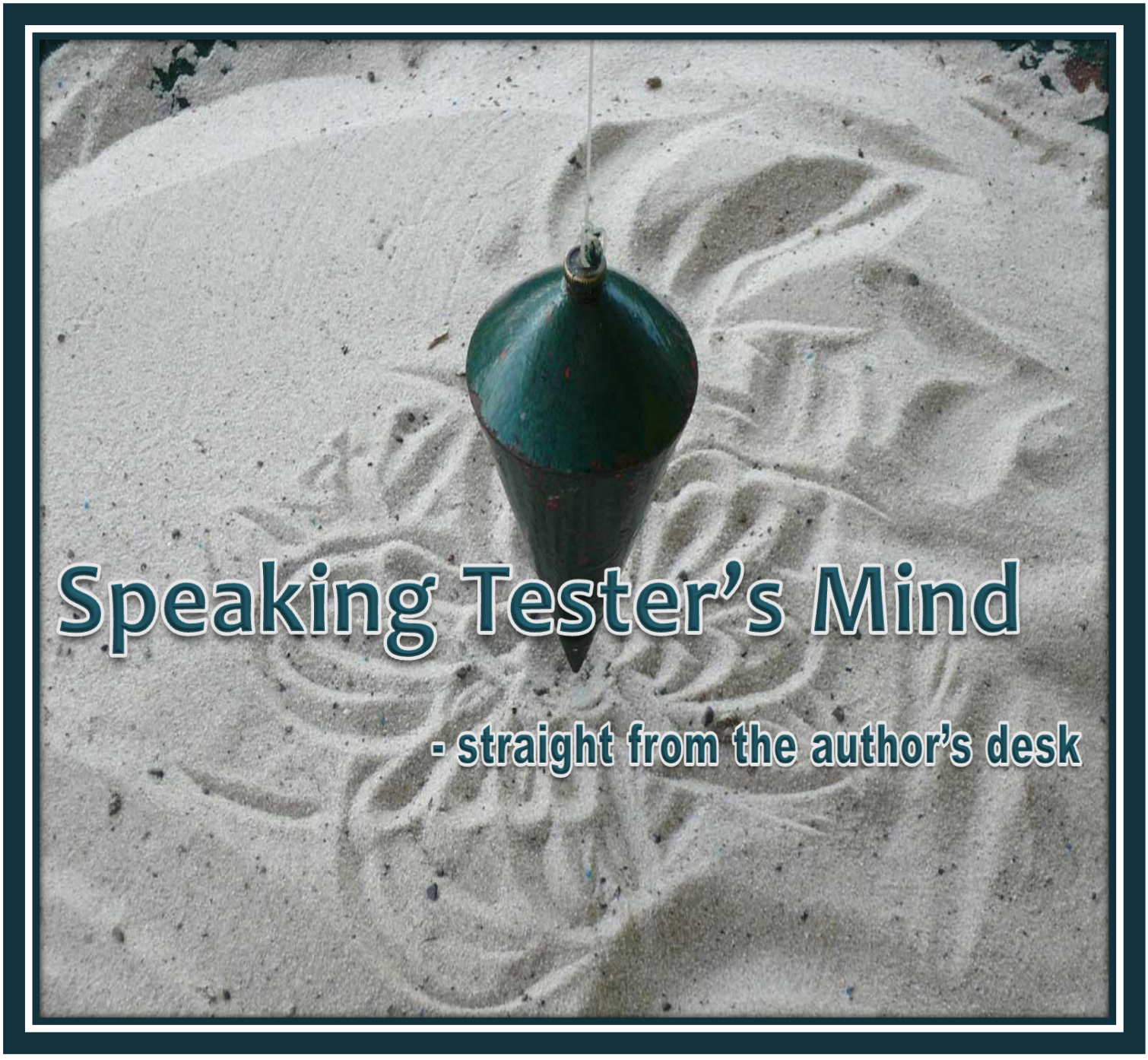
This is for everyone's sake. It gives the stakeholders the power, and thus the responsibility, for deciding what timeline everyone on the project is working toward. That's, in theory, part of their job. It's also good for the testers who, as Dawn Haynes explains in the clip above, shouldn't be making the decision to hit or miss a deadline.

Your job is to test the software as thoroughly as you can within the limitations handed down by the stakeholders, and then to provide as much information back to them as you can.

Don't Blindly Apply Any Practice

This is probably the most prominent and obvious pillar on which the context-driven ideology is based. As I stated in the introduction, best practices (including the ones listed in this article) will not work for every situation. Sometimes you simply won't gain much information from asking questions of project managers. Sometimes the project is in such dire straits that there really is no time to create a detailed test plan ahead of time. And sometimes it's better to stick with your guns and push back against stakeholders that are making a horribly detrimental decision. In the end, it's all about doing as much as you can with the information you have at any given time.

And it's exactly that kind of flexibility that will continue to elevate context-driven testers to the forefront of their field for decades to come.


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

Traditional methods of software development went over the waterfall in a barrel and smashed on the rocks below. The Agile Manifesto was written more than a decade ago. Agile adoption has worked wonders for software development. New products leap from concept to market faster than ever before.

Software updates are expected to roll out quickly into live products without disruption, the cloud is growing ever larger, and mobile technology is impacting heavily on how applications are developed. Feedback from the end user informs the design. Documentation and planning are sidelined in favor of flexibility and speed. It's a trend that has powered the app revolution, but it's so focused on developers that testing has been forgotten.

Clear out the old

The testing industry must evolve and adapt in order to keep pace with modern software engineering. New builds come thick and fast nowadays. It has become much tougher to estimate the required resources to properly test a project at the outset and, even if you could create an accurate estimation, it would soon be rendered obsolete.

It's not feasible to have small, fixed-size teams of testers covering an Agile project. As each new sprint introduces new features, the amount of work grows. Testers must check the new functionality, but they also have to verify bug fixes and complete regression testing. If you don't scale the team up as time wears on, or develop automated tests to reduce the workload, then things will start to slip.

Building a new approach

The Agile mindset in the developer community has given birth to countless methodologies, supported by books and dedicated software tools. The discussion and support is lacking in the testing industry and that needs to change. Testers need software tools that are up to the job. Test

management should cover the entire lifecycle of testing. An adjunct of the project management software designed for developers is not going to fulfill their needs.

The Agile age demands skilled testers equipped with tools that enable them to record tests step-by-step, link user stories and test cases, automate scripts where necessary, export and import bugs, and extract an overview of testing progress.

By developing exploratory testing skills and employing automation where it makes economic sense, testers can rise to the challenges of modern software development.

Employ Automation When Appropriate

The real value of testers is in their ability to test drive new features and validate bugs. You don't want them engaged in a regression testing slog. It means duplicating work and it's dull and repetitive for the testers. You won't get maximum value from your resources that way.

Automating regression testing can free up testers to focus where you want them. It's not easy, but with the right plan it can work. You can't create your test cases and scripts until the code is deployed. What you can do is record the testing process as your testers work through the new build and then use the steps captured to generate new test cases that can serve as the basis for automated regression testing on the next release.

Tester as end user

In the days of waterfall development, testers would have extensive documentation and requirements to pore over in order to create a detailed test plan. All the test cases would be prepared and ready to execute when the build arrived. That's often not possible with agile development. If the developers are going to adopt an agile mindset, then testers need to do likewise.

Exploratory testing can be employed to examine each new set of features when the build lands. Testers can record their steps and then edit them to create a solid base for regression testing. A core set of test cases can be fully scripted and automated. This process requires an evaluation of where the most value can be derived and that decision should be informed by what you know is coming down the pipeline from the developers.

For this to work testers must be included in the development process early and often. They need to be in Scrum meetings, they need to understand the user stories, and they must be empowered to contribute and ask questions. Testers can obviously learn a lot from developers on the project, but they should also be able to ask questions on the business side. If they can truly emulate the intended audience for the software then they can make a bigger contribution towards ensuring that it hits the mark.

Back To Index

Let's make a change

If we accept that the development landscape has changed irreversibly, then we can really focus on ways to empower testers. Adopt an Agile mindset, but apply it from a testing perspective. Seek out new processes and new tools that really deliver the functionality and structure testers need in order to add value in a timely manner. Encourage more communication and deeper involvement in projects, so that testers can emulate your end users accurately and help to ensure that expectations are met. Modern software development is still evolving and testing needs to evolve along with it.

Vu Lam is co-founder and Chairman of **QASymphony**, a leading provider of test management platforms for agile development teams. He was previously with First Consulting Group and was an early pioneer in Vietnam's offshore IT services industry since 1995.

He holds an MS degree in electrical engineering from Purdue University.

You may reach him at vulam@qasymphony.com.

Domain Knowledge!
How it helps you test better...

- by Juhi Bansal

I have been working as a software tester from the last 3 years now and have been associated with Public Sector Projects.

I am here not to teach you or not to make you cram some principles or theories which would make you a better software tester. My intent of writing all this is to share my experiences of working in Public Sector Projects and hope that you could connect to this.

Software Testing according to me is not testing an application just for the sake of clicking some random buttons and seeing if those work or not, or clicking different links on the page to see if they don't give any error. Software Testing has a deeper meaning than this which is actually testing or validating the application throughout to see if it really makes some sense and how will this be useful to our client and the end user.

When I started on my first Public Sector project, I was not at all aware of what the application means, why are we creating such an application and why me as a part of the testing team, validating it daily.
I faced a lot of problems while creating Test Cases because I did not have that domain knowledge. I was always stuck and not sure if I am writing a good test case. Over and above due to my lack of knowledge of the domain, I was never able to explain my test cases to my lead and always got confused thinking that "Okay. Maybe he is right and I am wrong". This made me think that what am I actually doing - just clicking Next, Continue, Back buttons on the page and not understanding the deeper meaning behind the rules; it may result in certain things not being tested and this may result in giving a 'not so good' application to the client.

I kept thinking about this and then I decided that I would first understand how this application really works, what the key components of our application are and how each component would affect the end user out there. While considering all this, I had in mind that I would want to think and test real life scenarios that may possibly occur out there with the end user and this in turn would make me confident in saying that "Yes, my application is well built and validated and will benefit the end user'

Understanding the functionality of the system is a very broad term and I would like to share my experiences regarding the same. When I started to learn about the application, my first teacher was Google. Yes, I started searching about the domain knowledge by searching and reading about the client

I was working for. This somehow helped me to connect to the domain on which I am working. I could find lots and lots of material plus this made me realize that our application would be affecting so many lives and if created and validating correctly, this would be beneficial for all the end users.

I started reading those material, started reading about their census, demographics. It made me realize the kind of people living there and as I earlier mentioned helped me to create real life scenarios that could be tested. I started reading about their current status and the no of people that would be affected if our system goes live. I remember one instance that left a lasting impression on my mind. It was a normal working day for me. I reached office and settled down. I started reading my assignments and somehow I was getting bored that day. I did not feel like working and then started to Google. I have the habit of bookmarking web-pages that I like and incidentally an already bookmarked page on Google Chrome opened and I was surprised. The page was about the status of the state for which I was working and it talked about how a well-integrated and tested system could help approximately 6 lakhs end users. The number surprised me. It made me realize that if I work hard and put my 100% in validating my system, then I would be helping those 6 lakhs people.

Understanding the functionality of the system or the domain not only helps to validate better but also makes you confident when you report your statuses to your leads. It helps you to think out of the box, makes you think about different scenarios that may happen in the real world.

Initially I thought that the scenarios which we need to test will be very different and it will be a difficult task to think about scenarios and test the. But as and when I started understanding the system and the domain in which we are creating the application, I understood that it was not at all difficult. All you need to understand is the target audience which would be using your application and the purpose of them using the application.

During the course of my being a tester in Public Sector Projects, I have seen people testing the application just for the sake of completing their assignments for the day and not bothered about what is the main intent of validating a particular functionality or component. The only point of being a tester is to make sure that you bring out the best in the system and this will happen only when you understand the system well, understand the domain and then try to break the application.

As the father of testing – Glenford J. Myers has rightly said – Tester wins only when he breaks the application not when a particular functionality works as expected. Of course the main intent is to make the application work as expected and I also understand the fact that exhaustive testing is not possible but we need to make sure that before we hand over the application to the client we are pretty confident that we have tested it well. Tested it well does not mean testing all the given test cases and marking them as pass and done. NO. Tested it well means that we have tested enough permutations and combinations and we are sure that the application would not break. Of course no software is defect free but we need to ensure that the public sector projects, one which cater to the basic needs of the people is at least 90% defect free and this can be done only when we have people in testing team who have that mindset of breaking the application by trying out different kinds of scenarios.

I think there are a lot of ways to learn and understand the application/domain

- Reading the documents for that particular client and understanding the needs of the end users
- Talking to the Business Analysts
- Talking to the development team
- Studying the Functional Documents
- Sharing your knowledge with others
- Documenting details for future reference

- Curiosity to learn about new things
- Giving attention to the details

Lately my lead taught me a very good point which I would like to share with you all- This also talks about understanding the domain. When a software tester is given a defect to validate, what testers do is – read the defect, validate the scenario which was failing and mark the defect as PASS.

This is an incorrect approach to testing. What actually testers are required to do is, understand the defect, and talk to the developer what was the fix made of course not technically but on a high level as to what functionality was fixed and how can it have an impact on other functionalities. Once done, write down the fix and now try to brainstorm under what scenarios will the user encounter that functionality plus you need to think that what other functionalities could have been impacted due to this fix. This will give you dual benefits – one you will test all the scenarios which would test that particular functionality which failed earlier and you will do regression testing to make sure that somewhere something else did not break while fixing this defect.

I can share with you the advantages I got when I gave importance to the domain knowledge. There was a small project for a state where I was testing for the Development team and when it went to the testing team, they could not find more than 10 defects. This was my first achievement. I got into the second Public Sector Project where I logged about 400+ defects. Currently I am working on a project where I already have added 500 valid defects in my kitty. Apart from this, I got an opportunity to make changes in the design of the system based on the domain knowledge I have.

So you see how important is it to understand the domain in which you work.

Hope you enjoyed reading my experiences. I have a lot to share about other experiences as well but for now I think I should be done.

See you soon.

Juhi Bansal has been associated with Software Testing since 3 years. During this period she has held positions that involve every aspect of the software development life cycle from writing test cases to executing them, understanding and validating change requests to business analysis. She has demonstrated the ability to engage in decision making discussions related to business processes and have the expertise in mentoring other team members. She contributes extensively to the other projects in terms of functional knowledge. Juhi is confident under pressure and highly resourceful to drive successful client solutions. I has a strong Self-driven/Proactive approach in the workspace and go the extra mile to delight customers. She is interested in attending software testing conferences held across India and learn about the best practices of testing.

Apart from this she makes sure that she reads testing magazines and become a part of on-line testing communities to learn more from other's experiences. She is keen on to enhancing these skills.

Back To Index


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

The current enthusiasm for Big Data is intriguing, almost as fascinating as the subject itself. I wish there had been a similar level of interest back in the mid and late 1990s when I worked with huge insurance management information (MI) data warehouses as a development team lead, project manager and test manager.

This was highly complex and demanding work, and life would have been easier if more people in IT had had a clearer idea of what we were up to. The trendy work then was all real time database systems and the early web applications. The attitude to our data warehouse work was summed up by a newly arrived manager who was given a briefing about what we were doing; he said, "So, you're working on batch legacy systems?"

Well, the work was batch, but in financial services that's often where the really complex, intellectually demanding IT work is done. And yes, we were dealing with old applications, but this was a strategic programme to extend the old applications to add vital new functionality.

"It's not enough to know we lost money – we have to know why!"

Our mission was to standardise the various sources of MI within the company, pulling them together into a system that could be used both by insurance managers and the statisticians who monitored profitability and set the premiums. This required many new interface applications to take raw data from the source underwriting and claims systems into MI data warehouses for subsequent processing by a new front-end system that the insurance managers would use.

The statisticians would crawl all over the new data warehouses building a detailed understanding of what risks we were facing, and how they should be priced. The managers would look at the results of the predefined analyses that reduced the vast amounts of messy data to clear and simple analyses of profitability.

It is vital for an insurance company that it understands its portfolio so that it not only knows which customers are profitable, but also why. Otherwise the insurer will gradually lose the profitable business to rivals who are better informed and can set appropriate rates. The remaining customers will be the bad risks. Accurate and timely management information is therefore a matter of business survival. "It's not enough to know we lost money – we have to know why!"

Making the bricks for the data warehouse

All of our insurance systems were designed for processing underwriting and claims. The data was therefore not held in a form suitable for MI. Converting historical transaction data into the right form was a surprisingly difficult and complex job. Basically the reformatting entailed matching the premium income and claims payments with the factors that earned or lost the money; e.g. for a given package of cover we sold to a customer the company earned £x. This information could then be used as the basic building brick for the sophisticated analyses required by the business.

We had to reformat the historical data and also set up feeds that would take the ongoing processing data and convert it.

My first draft of this blog got bogged down in the technicalities of insurance finance. Once you get sucked into trying to explain the significance of the differences between written and earned premiums, and between incurred and occurred claims then it's hard to know where to stop. Trying to keep it simple just leaves the reader baffled and doesn't convey the massive practical problems involved in converting the data. Explaining the issues precisely is a boring turn-off.

So I've ditched the financial detail and I'll try to concentrate on the bigger, more interesting issues.

Big Data = big problems

Firstly, and most obviously, Big Data meant big problems. When we started working with files that were 10, 50 even 100 times bigger than the files we were used to it became clear that the old ways wouldn't work. Run times and disk space allocations had to be carefully calculated. Batch suites had to be very carefully designed. Important though this was it wasn't our toughest challenge. Our biggest problem by far was testing.

Traditional linear techniques suck

This was the time that I really ran smack into the fact that traditional, linear techniques suck. They suck particularly badly when you're dealing with a highly uncertain situation. Uncertainty is the reality in software development, and that simple truth was a factor I underestimated massively when I planned and led the first of my data warehousing projects. Build it then test it was a plan for disaster.

The whole point of our development was to provide the business with information that was not otherwise available. If the information could have been provided more easily, by some alternative means, then it would already have been done. There was therefore no readily available oracle against which we could test.

Traditional test scripts were irrelevant. How could we sensibly draw up scripts with predicted results based on our input when we had no real idea of the potential problems? We didn't know what we didn't know! I planned the project based on what the source systems should have been doing, what the

source data should have been, and I allowed for the problems that we should have been able to foresee. How naïve!

Across time, not just at a point in time

We built the system and only then did we start seriously testing it. Sure, the programmers had done careful unit testing. But what we hadn't allowed for was that in building a data warehouse that covered a decade of processing, we needed accuracy and consistency across time, not just at a particular point in time.

Successive versions of a motor policy might be entirely accurate and consistent with accounts and claims data at a particular point in time. That didn't necessarily mean that these successive versions were consistent with each other, at least not to the level of detail and accuracy that we required.

Numerous changes had been made to the source systems, none of which had affected the integrity of processing, but all of which had subtle, but cumulatively massive, effects on the integrity of the MI that the data could provide. Also, trivial bugs that might have been ignored, or not even noticed, in the processing system could have a much more significant impact on the potential MI.

We'd always known that accuracy and consistency were crucial, but we hadn't grasped just how much more complicated and difficult the problem would be when we introduced the extra dimension of time.

The big lesson I learned was that traditional techniques condemned us to building the application in order to find out why it wouldn't work!

We managed to dig ourselves out of that hole with numerous coding changes, some frantic data cleansing and a ruthlessly dramatic redesign that entailed axing half of the system and replacing it with a cloned, and then adapted, version of the surviving part.

That approach was clearly unacceptable. So for the following MI developments I adopted a more practical, efficient and effective approach. There could be no artificial distinction between the build and the testing. What was required was a form of test-driven development. There were two main strands to that.

Lesson 1 – tester, know your data!

Firstly, before the development could start we had to explore the source system and its data. We had to do it thoroughly. I mean really, obsessively thoroughly, not just quick scans to try and reassure ourselves that our optimistic assumptions were valid.

We would crawl though the source data to understand it, to identify patterns and relationships that we could exploit in testing and problems that would later screw up the statistical analyses. We had to find the patterns that existed not just horizontally across all the data at a particular moment in time, but also the patterns that unfolded over time.

It was amazing how often the data failed to match the way the system was assumed to work, and how the patterns would appear then evolve over the years. This knowledge was obviously vital for the build work, but it was also priceless for testing.

The lack of readily available test oracles meant that any relationships that held true over time, or over a large number of records, gave us something to hook our testing on. E.g. for a given policy the

written premium on an individual transaction bore no necessary relation to the earned premium. It could even be negative. But over the full length of an insurance contract the sum of the written premiums must equal the premium that was earned.

We'd go round in circles learning more and more about the data, applying new insights, trying out new ideas till we had a load of relationships and rules. These rules were a mixture of business rules, rules that could logically be inferred from the data, and possibly quite arbitrary rules imposed by the design of the source systems. Such rules might have been arbitrary and of no business significance, but breaching them would mean we'd done something to the data that we'd not meant to and didn't understand. We could get guidance, not requirements, from the users to get us started. However, that guidance consisted of what ought to be happening in the present, and was therefore of limited value.

We'd then build these rules into the processing. Basically we'd design the processing around them. In live running these checks would flag up any deviations. Serious discrepancies meant the run would stop and some poor soul would get a phone call in the middle of the night.

Lesson 2 – build it so you can test it

The second strand to the development testing was also tied into the design. It was important that the batch suites were broken up into discrete stages that could be run in isolation with meaningful, testable results at the end of each stage. We could then step slowly through a whole suite, testing the results at each stage. The processing would have been far more efficient if we'd lumped more into each stage, ideally processing each record only once, and doing everything necessary with a single access.

We had our fingers badly burnt when we took that efficient approach with the design of the first application I was talking about. It meant that significant defects could be a nightmare to debug. There was a trade-off between the strain we were imposing on the batch processing window on the one hand and the significant cost of testing, fixing and retesting and even redesigning. Efficiency was important, but obsessing about it was a false economy. Testability had to be the most important factor dictating our designs.

Not real testing?

At the time I didn't consider that what we were doing was real testing. It was what we had to do in the circumstances. Real testing was all about scripts and test cases, and that was very much the view of the testing specialists at the company. When I actually moved into test management and thought more deeply about what testing meant I realised how wrong I'd been to dismiss our work as "not real testing", but how right I'd been to insist that we should do what fitted the problem, not what fitted the development and testing standards.

I've been leafing through performance appraisals and post-implementation reviews from the period. One appraisal said "the project required considerable business analysis work where James displayed a special aptitude to get to the bottom of complex situations".

Real testing? I think so.

James is a self-employed testing consultant. He has 30 years experience in IT. He has worked as a test manager, IT auditor, information security manager, project manager, business analyst and developer. He is particularly interested in the links between testing, governance and compliance. He is a member of ISACA, the Information Systems Audit and Control Association. James spent 14 years working for a large UK insurance company and then 9 years with IBM working with large clients in the UK and Finland.

He has been self-employed for the last 7 years.

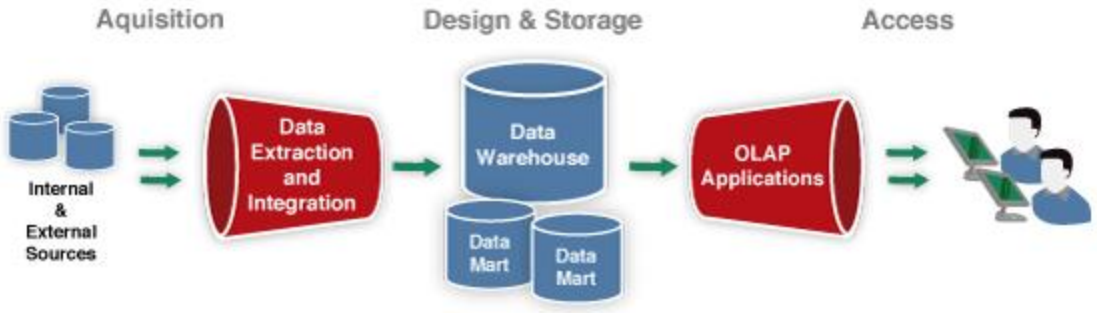
Delivering Data warehouse Quality – An Obligation to Your Business

- by Wayne Yaddow

Data warehousing for business intelligence analytics and "big data initiatives" continues to gain significance as organizations become more fully aware of the benefits of decision oriented data warehouses. However, a key issue, with the rapid development and implementation of data warehouses, is that data quality defects are often injected during the multiple lifecycle stages of the warehouse.

A primary requirement is for an efficient data warehouse (DWH) system process that reliably extracts, transforms, cleanses and loads data from source systems on a 24 by 7 basis without impacting overall performance, scalability or reliability.

In this article, we present new ideas on a "beginning-to-end" data warehouse life-cycle quality process.



Eliminating data quality errors: Injections of data quality problems occur during all phases of data warehousing: 1) data warehouse modeling and schema design, 2)ETL (extract, transformation, loading) design and coding, 3) integrating data from varied sources, and 4) running the ETL processes for data-staging, cleaning, and loading. It's common, although undesirable, for problems and defects to emerge when populating the warehouse.

Data testing is often planned for the later phases of data warehouse projects. However, most QA professionals agree that establishing a successful test planning and execution effort for the early project phases is one of the keys to success. As with other software development, the earlier data errors are detected, the less expensive it is to find and correct them. Besides that, planning early testing activities to be carried out during design and before implementation gives project managers an effective means to regularly measure and document the project progress state.

QA efforts should begin early: Since quality of a DWH can be measured best with reference to a set of data requirements, a successful testing process begins with the gathering and documentation of end-user data requirements. As most end-users data requirements are about data analysis and data quality, it is inevitable that data warehouse testing will focus primarily on the ETL process on the one hand (this is sometimes called back-end testing), then on reporting and OLAP on the other (front-end testing).

After gathering requirements, analysts develop conceptual, then detailed schemas to represent user's needs as an important reference for testing. Designers are responsible for logical schemata of data repositories and for data staging definitions that should be tested for efficiency and robustness.

The data architecture and model is necessary as a blueprint for any data warehouse. Understanding these artifacts and discovering potential defects help the QA team to comprehend the bigger picture of a data warehouse. The data model aids comprehension of the methods used for the key relationships between the major data sources. The relationship hierarchies and the depth of data throw light on the complexity of transformation rules.

In order to gain greater value from the QA team, it's recommended to include them for quality assessments in all phases of data warehouse design and development and testing. Figure 1 shows typical DWH project testing categories. Representative validations are shown associated with four significant value QA tasks including a) *data testing, b) GUI /business rule testing, c) database / schema testing, and d) performance testing*. Reviews, verifications, recommendations for improvement are among QA team contributions that aid in early removal of defects.

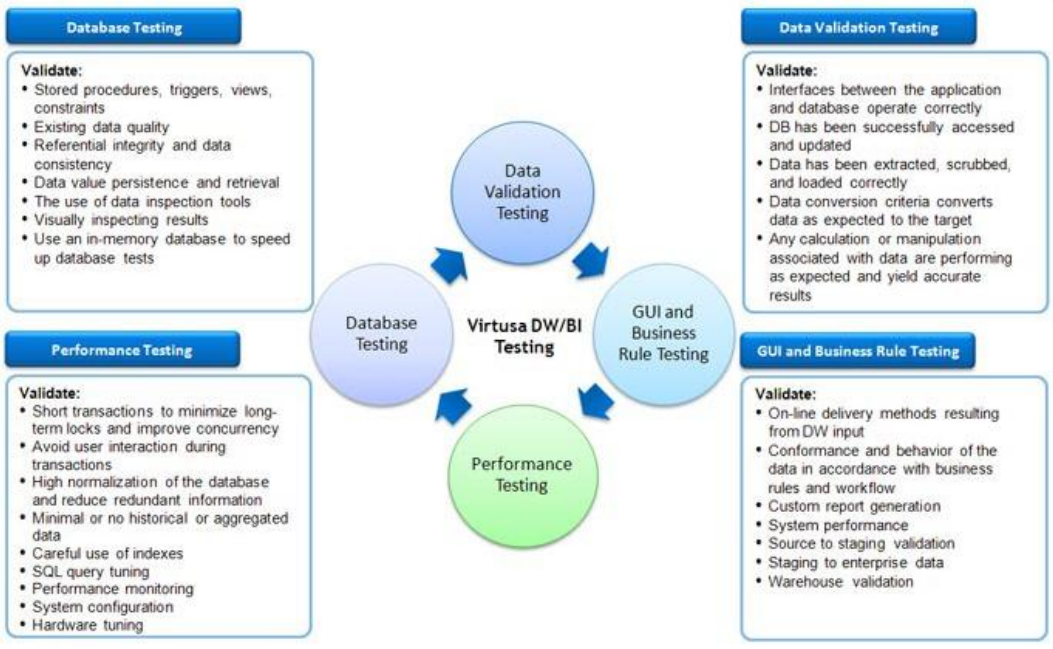


Figure 1. Categories of data warehouse testing to discover quality issues (graphic courtesy of Virtusa Corp.)

QA team contributions during the DWH project lifecycle

Following are sample contributions and benefits from the QA team during the DWH development lifecycle:

Planning for data integration and the ETL (data model, low-level DWH design)

- Testers gain an understanding of data to be reported by the application (e.g., profiling) and the tables upon which BI and other reports will be based
- Testers review and understand the data model – gain understanding of keys, flows from source to target
- Testers review and become familiar with data LLD's and mappings: add, update sequences for all sources of each target table

Planning for ETL verifications

- Testers participate in ETL design reviews
- The QA team gains an in-depth knowledge of ETL workflows / sessions, the order of job executions, restraints, transformations
- Testers develop ETL test scenarios and distribute for reviews

Assessing ETL run and diagnostic logs: session, workflow, errors

- After ETL's are run, testers use checklists for QA assessments of rejects, session failures, errors
- Review ETL workflow outputs, source to target counts
- Verify source to target mapping docs with loaded tables using TOAD and other tools
- After ETL runs or manual data loads, assess data in every table with focus on key fields (dirty data, incorrect formats, duplicates, etc.). Use TOAD, Excel tools. (SQL queries, filtering, etc.)

During DWH design and development, testers should plan and organize for the following DWH quality categories:

Data completeness: Ensuring that all expected data is loaded

Data transformation: Ensuring that all data is transformed correctly according to business rules and/or design specifications

Data quality: Ensuring that the ETL system correctly rejects, substitutes default values, corrects or ignores and reports invalid data

Performance and scalability: Ensuring that data loads and queries perform within expected time frames and that the technical architecture is scalable

Integration testing: Ensuring that the ETL process functions well with other upstream and downstream processes

User-acceptance testing: Ensuring the warehouse solution meets users' current requirements and anticipates their future expectations

Regression testing: Ensuring that current functionality remains intact each time a new release of code is completed

Testing cannot guarantee that there will be no data errors. There are too many combinations and permutations, so it is not practical to test each one. However, by joining forces with BA's, DB designers, developers and ranking the types of errors as suggested above, DWH projects will avoid wasting time on creating test scripts and test scenarios for less important possibilities and not having time to create test scripts and test scenarios for possibilities in which errors could significantly diminish or destroy the value of the data warehouse to the users.

To be continued in next issue...

Wayne Yaddow is a computer testing professional. He worked with IBM as a Z/OS mainframe operating system developer and tester for fifteen years. After IBM, he continued his career as a QA consultant, working primarily in the financial industry in NYC with firms such as Standard and Poor's, Credit Suisse, Citigroup, and JPMorgan Chase. Wayne focused on business intelligence, data warehouse, data migration and data integration testing projects.

As a contributing author to Better Software, TDWI Business Intelligence Journal and a participant with The Software Testing Professional Conference (STP), he has gained a reputation as a well-informed database and data warehouse quality assurance analyst. Most recently, Wayne teamed with Doug Vucevic to write the book, "Testing the Data Warehouse", 2012, Trafford Press.

Dealing with Asynchronous

a series by Jim Holmes

Intro

In my previous articles in this series I've discussed why you should look at UI automation, how to select tools, and how to handle element locators. I had to take a month off while changing jobs; however, here we go with the next installment in the series—this one on helping you avoid tearing out your hair over odd timing issues in your tests.

Have You Seen This Before?

Here's a scenario very common to teams new to, or even moderately experienced with UI automation: A test script is created and validated on the tester's/developer's system. It works, so it's checked in to source control and added to the regular automation suite for regular execution.

Later that day the new test runs in a different environment (different execution systems, different application servers, etc.) and the test fails. The error message may be something rather strange indicating a target element or piece of data on the page couldn't be found. You re-run the test in your local environment and the test passes.

A cycle of tweaking timing and re-testing begins as you and your team try to get the tests stable. Frustration mounts, wasted time increases, and occasionally teams abandon automation efforts completely due to the lack of value.

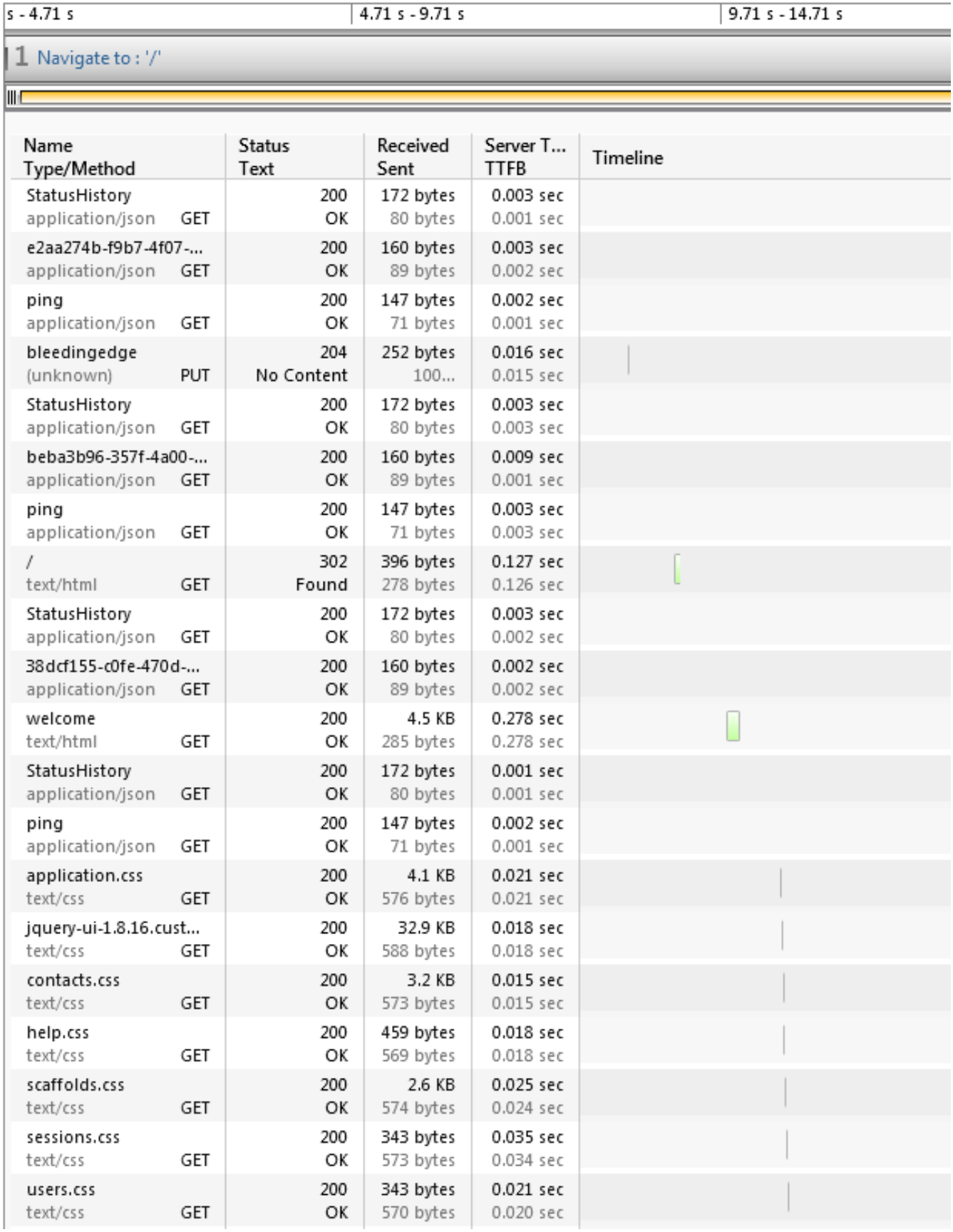
Why These Failures Happen

Timing problems like this occur because of differences between a browser loading an entire page versus updating a part of the page. (The same notion holds for desktop or mobile application views.) Understanding the differences is a critical step in building stable UI tests.

Understanding the Page Load Cycle

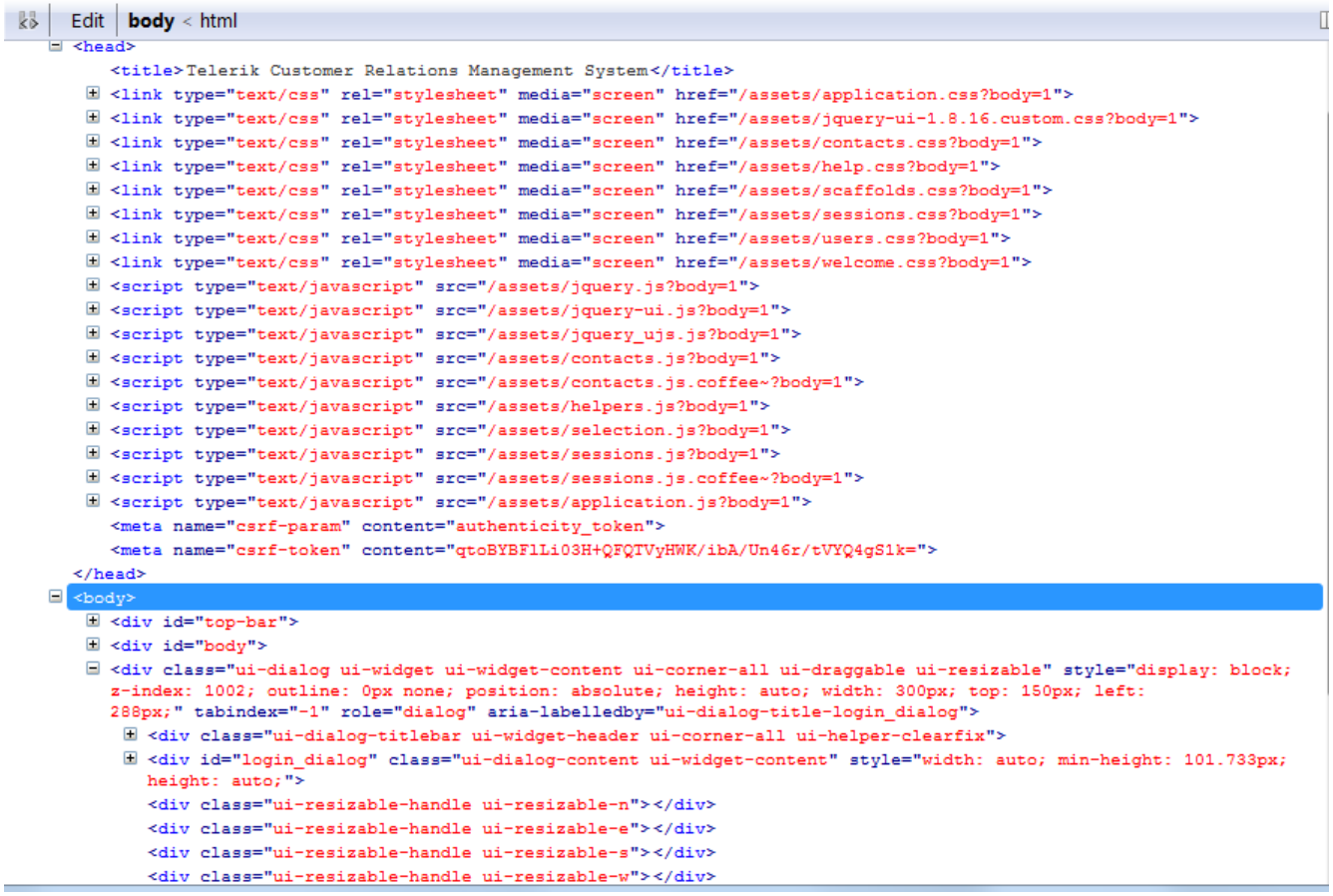
Loading or refreshing/reloading pages in a web browser is a blocking action. This means the browser stops until the page is finished loading. Breaking down the actions involved in a page load is very helpful in understanding what's going on behind the scenes.

When a browser navigates to a new location a series of conversations kicks off between the browser client and the web server. The browser asks the server what it has available at that particular URL. The server responds by sending all the static items making up that page: text, images, CSS, JavaScript, etc. The image below is from Telerik's Test Studio and shows parts of this conversation. (Many other tools give you the same sort of visual representation, too.)



Name Type/Method	Status Text	Received Sent	Server T... TTFB	Timeline
StatusHistory application/json GET	200 OK	172 bytes 80 bytes	0.003 sec 0.001 sec	
e2aa274b-f9b7-4f07-... application/json GET	200 OK	160 bytes 89 bytes	0.003 sec 0.002 sec	
ping application/json GET	200 OK	147 bytes 71 bytes	0.002 sec 0.001 sec	
bleedingedge (unknown) PUT	204 No Content	252 bytes 100...	0.016 sec 0.015 sec	
StatusHistory application/json GET	200 OK	172 bytes 80 bytes	0.003 sec 0.003 sec	
beba3b96-357f-4a00-... application/json GET	200 OK	160 bytes 89 bytes	0.009 sec 0.001 sec	
ping application/json GET	200 OK	147 bytes 71 bytes	0.003 sec 0.003 sec	
/ text/html GET	302 Found	396 bytes 278 bytes	0.127 sec 0.126 sec	
StatusHistory application/json GET	200 OK	172 bytes 80 bytes	0.003 sec 0.002 sec	
38dcf155-c0fe-470d-... application/json GET	200 OK	160 bytes 89 bytes	0.002 sec 0.002 sec	
welcome text/html GET	200 OK	4.5 KB 285 bytes	0.278 sec 0.278 sec	
StatusHistory application/json GET	200 OK	172 bytes 80 bytes	0.001 sec 0.001 sec	
ping application/json GET	200 OK	147 bytes 71 bytes	0.002 sec 0.001 sec	
application.css text/css GET	200 OK	4.1 KB 576 bytes	0.021 sec 0.021 sec	
jquery-ui-1.8.16.cust... text/css GET	200 OK	32.9 KB 588 bytes	0.018 sec 0.018 sec	
contacts.css text/css GET	200 OK	3.2 KB 573 bytes	0.015 sec 0.015 sec	
help.css text/css GET	200 OK	459 bytes 569 bytes	0.018 sec 0.018 sec	
scaffolds.css text/css GET	200 OK	2.6 KB 574 bytes	0.025 sec 0.024 sec	
sessions.css text/css GET	200 OK	343 bytes 573 bytes	0.035 sec 0.034 sec	
users.css text/css GET	200 OK	343 bytes 570 bytes	0.021 sec 0.020 sec	

The resulting Document Object Model (DOM) hierarchy for this particular page looks like this:

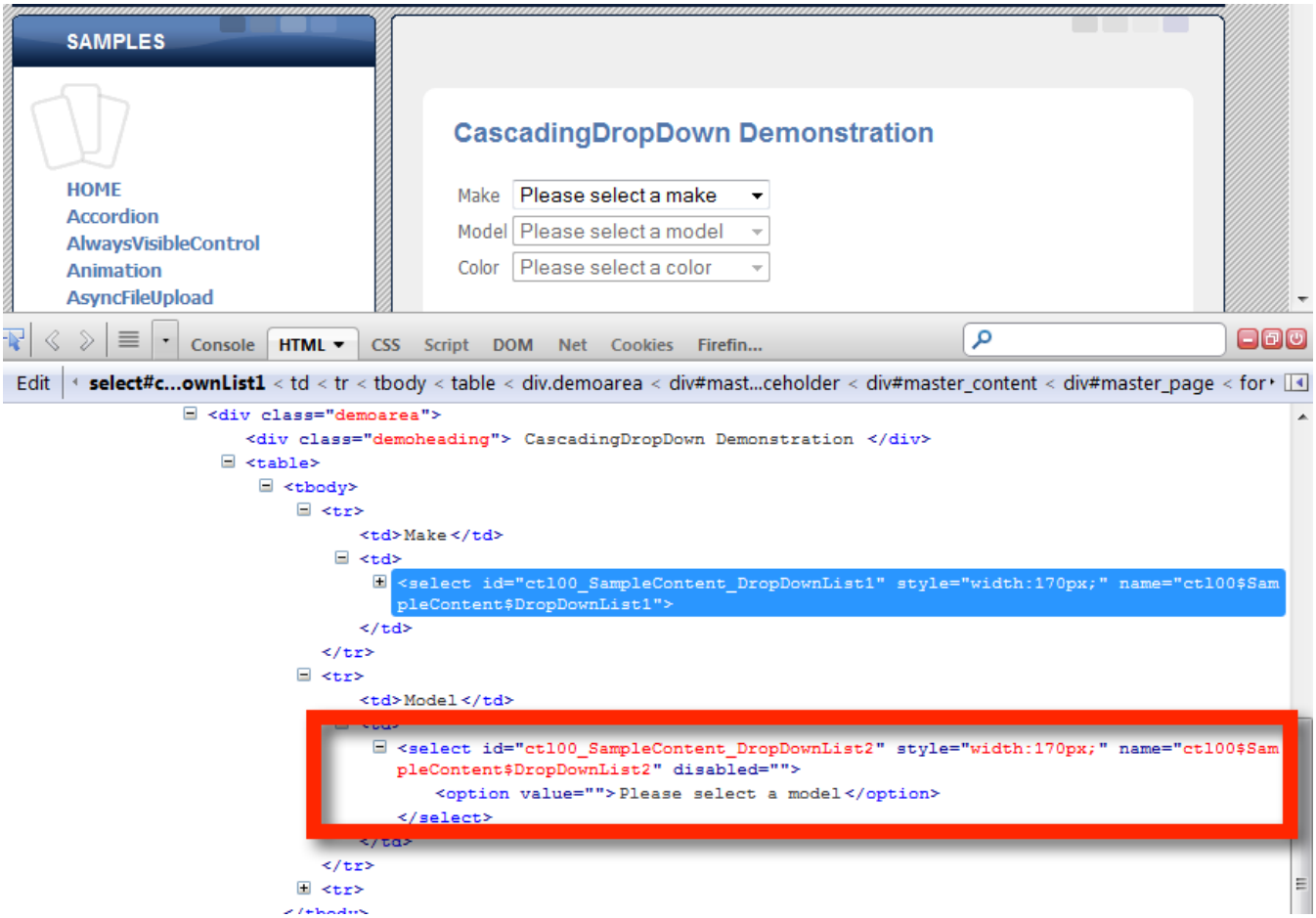


It makes sense that a browser would stop all other actions while these pieces are being received and assembled. Every browser sets a metaphorical Stop! sign on the page's DOM as this is occurring. The flag isn't cleared until all the chunks of data are received and assembled/rendered on the page. Tools such as web automation drivers check the status of this flag before moving on to their next actions.

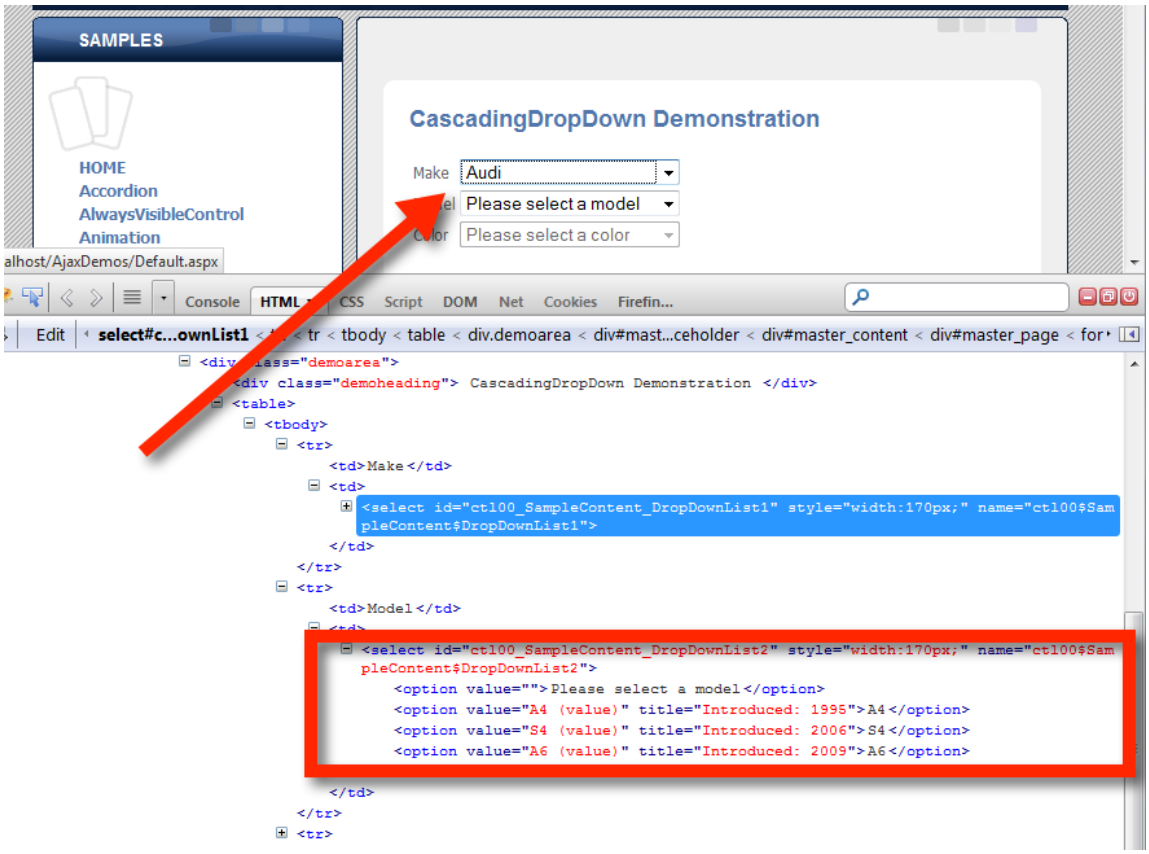
AJAX and Client-side Tools: A Wrench in the Works

Loading and reloading a page is an expensive action. It's time consuming for the users and can directly impact your organization's costs if you're paying for bandwidth usage. Moreover, it's senseless to incur that hit when only a small part of the page has changed. AJAX is one technological approach that lets the client call back to a server and pull back only the data that's needed for the page.

Automation problems arise in these situations because these calls are asynchronous—they don't block the page via a page load/refresh—they're specifically designed to *avoid* doing that in order to skip the time and bandwidth consuming process. Microsoft's AJAX demo pages' Cascading Drop Down example shows this issue perfectly. In the image below, the Make options are populated, but the Model aren't.

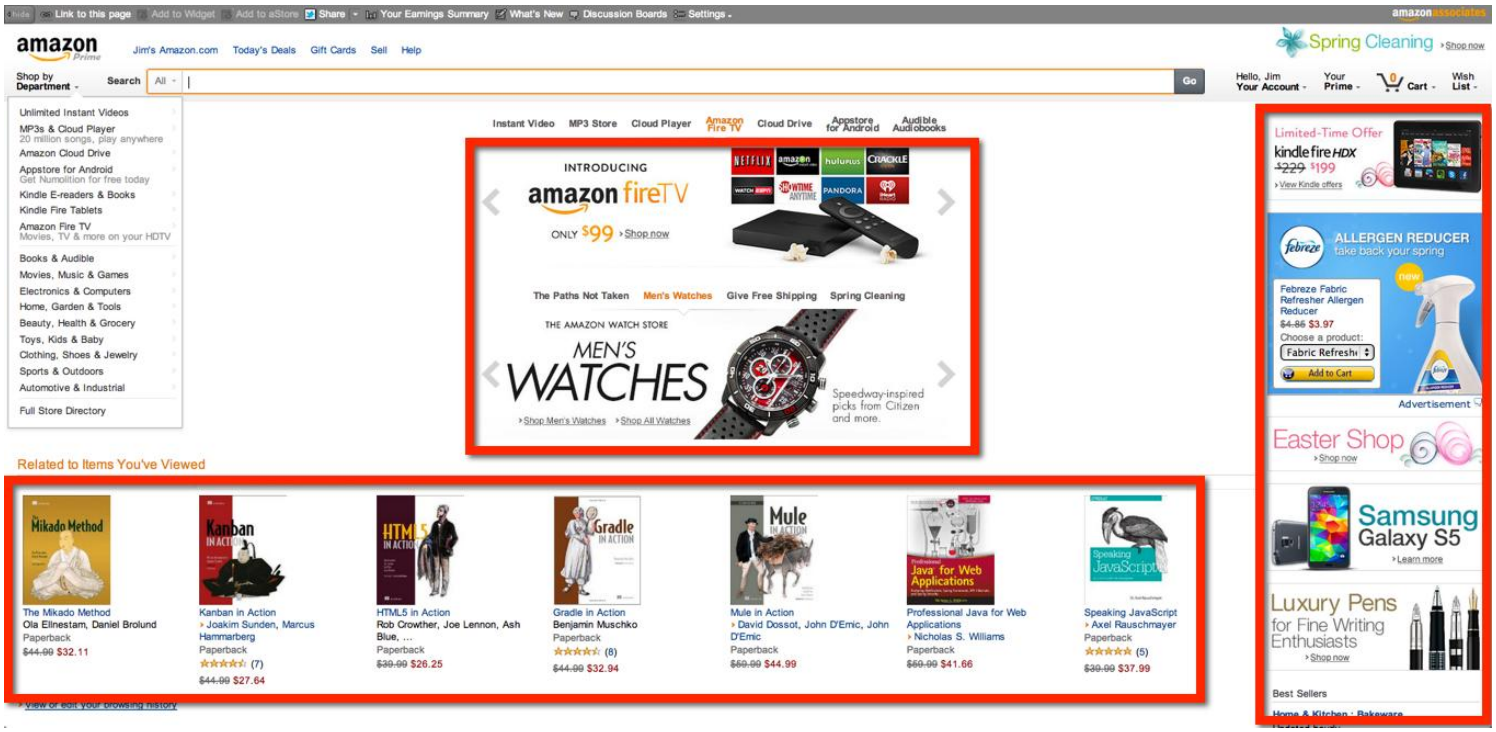


As a user selects a Make, there's a callback to the server, and a new set of option items are added to the DOM:



Because this is an AJAX call, there's no page reload/refresh. As a result, there's no change to the "stop sign" on the DOM. Browser automation tools don't know to stop while the call is in progress. As a result, test failures pop up because of the delay as the server callback completes.

The example above is fairly simplistic—it's just a series of one-at-a-time calls. The real world's much more complex. Think about Amazon's homepage which has multiple concurrent AJAX calls going on simultaneously!



Even frameworks like JQuery, ember, and other client-side tools can also cause page updates without altering the page's stop sign. Think about pages that cause new controls or content to appear based on the actions of a user. There might not be a server call involved; the UI may change based purely on state in the client!

Staying Sane With Reliable Solutions

Too often teams rely on bad workarounds for asynchronous operations. Falling back to scattering Thread. Sleep (30000) statements through a test suite. These hardwired pauses work, but they're the completely wrong approach to solving the problem. These fixed delays add up to significantly slower test suites, especially when you're talking hundreds of tests. Simple math shows 30 seconds of delay in 1,000 tests is 30,000 seconds. That's over eight hours just in manual delays!

Manual pauses are a horrible way to solve timing issues in your tests. Instead, automated tests should rely on waiting for an explicit condition in the system—nearly always the condition that's needed for the next step in the test.

Explicit Waits to the Rescue!

Rather than relying on hardwired manual delays, good automation suites leverage their toolsets' support for dynamic polling that pauses only as long as needed until an explicitly defined condition is met. These patterns are supported by every automation toolset in one form or another.

Why Wait

Dynamic conditional waits have several advantages over manual pauses (Thread.Sleep(), eg). First, you've explicitly set the condition to wait for. Clarity of intent in software is critical: people looking at the script later on understand exactly why the script is pausing. There's far less chance for misunderstanding the script's intent.

More importantly, the wait isn't hardwired, it's dynamic. That means your scripts will always adjust to varying conditions in your environment. Is the server bogged down with other work today? You won't run into failures because your timing was set up for yesterday's server speed. Network crushed by traffic? Again, no worries. Your explicit waits' dynamic timeout periods will flex and cover you in non-extraordinary circumstances.

Finally, you won't be adding hours to your test suites' execution times because the dynamic waits take as long as needed for the condition to be met (or the timeout expires). Your suites will remain running as quickly as possible.

How to Wait

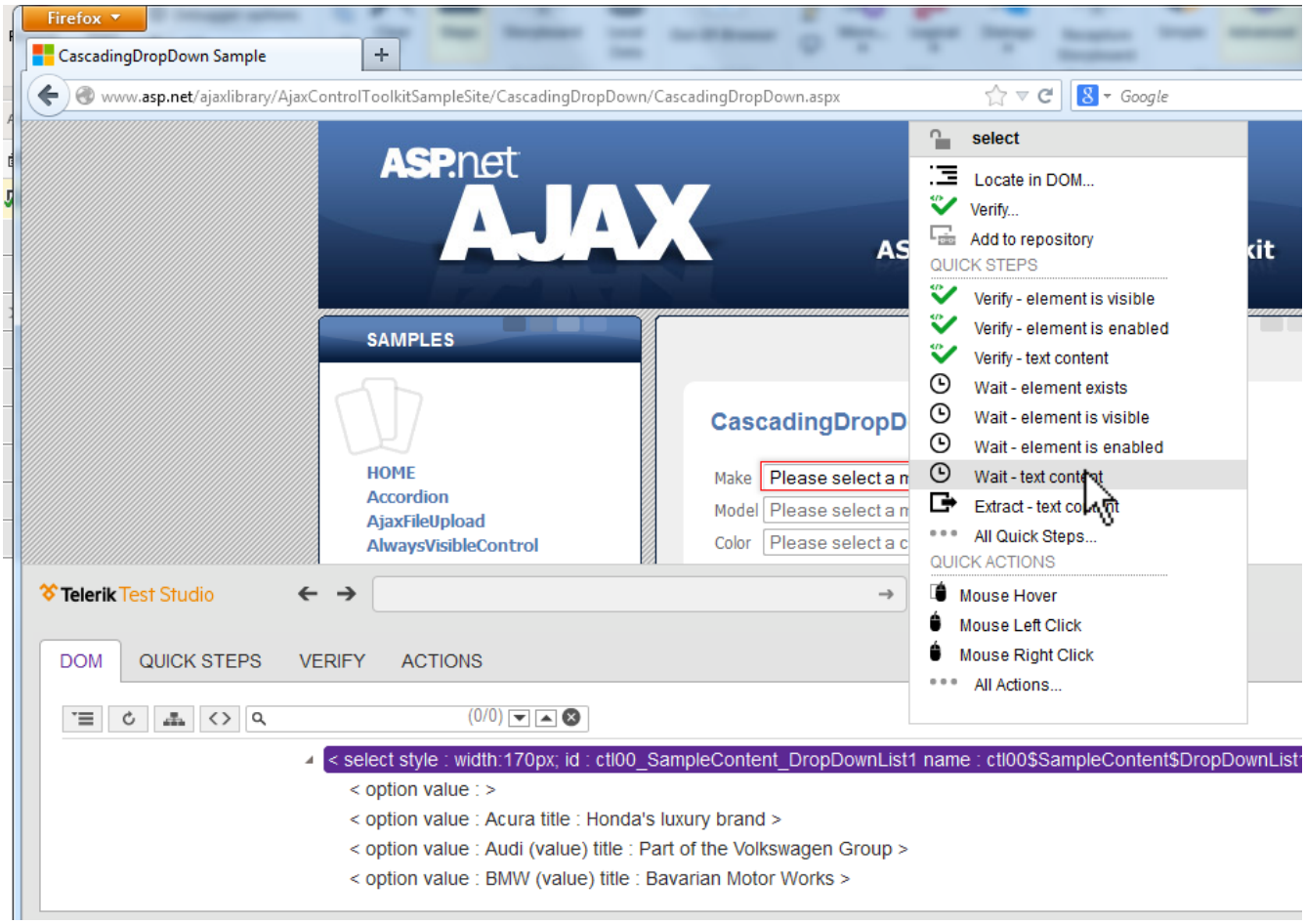
Conditional wait statements (also called explicit waits) generally work like this:

- A condition to wait for is defined. The condition might be a specific item being loaded into a menu, a control appearing on the page, or text appearing in a search results box.
- Actions are taken on the page firing off the server callback or other dynamic action
- The explicit wait statement/step/action is reached, and a check is performed to see if the condition is met
- If the condition is met, the statement completes and the script moves on
- If the condition is unmet, the script sleeps for a moment
- Repeat this loop until the condition is met, or a timeout is reached

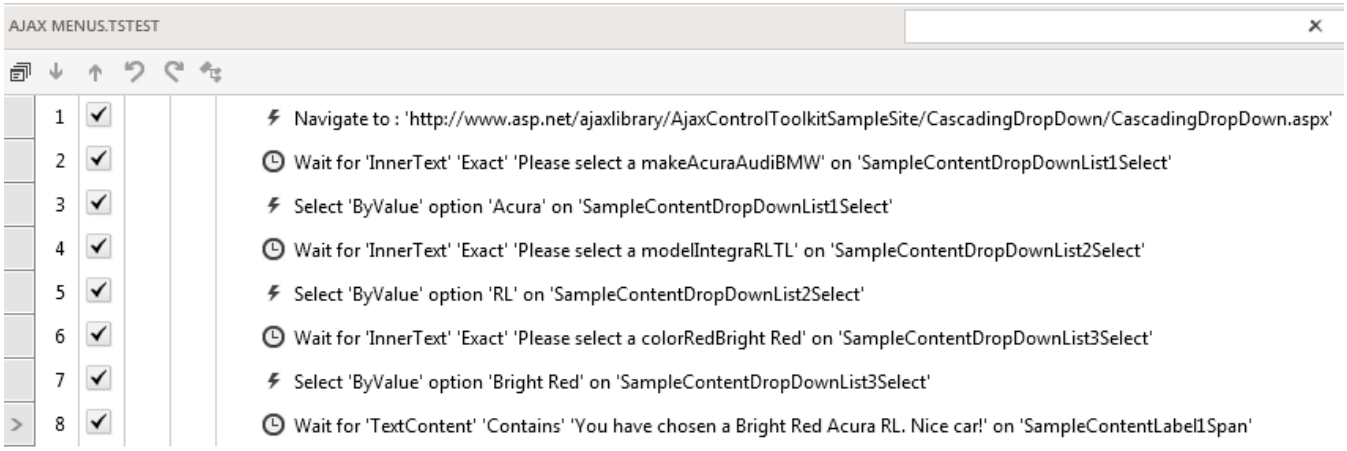
The exact mechanics of explicit waits varies by the automation tool you're using. Selenium WebDriver provides the WebDriverWait class. Below is an snippet in C# showing how to handle the AJAX cascading menus shown earlier:

```
WebDriverWait wait = new WebDriverWait(browser,

wait.Until(ExpectedConditions.ElementExists(

By.XPath("id('ctl00_SampleContent_DropDownList/option[text()='" +

menu_item_to_wait_for +

"']")));
```

Other tools handle explicit waits in different manners, but the concept's the same. Telerik's Test Studio, and other similar tools, give you the ability to add in explicit waits as you're crafting your tests.



The resulting tests look similar to this:

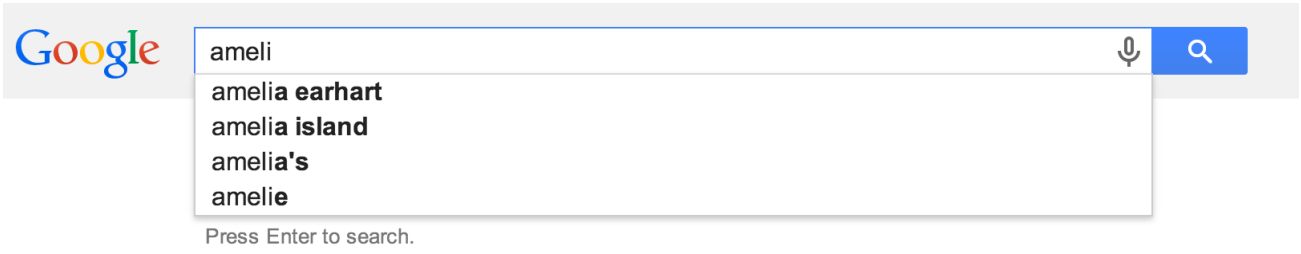


What to Wait For

We've quickly run through the mechanics of how an explicit wait works, but we've left off a tricky piece of the puzzle: *what condition* you should wait on. This actually isn't all that tricky a concept. The easy, rock-solid trick is this: wait on the exact condition you need for the next step in your automation script.

Take the cascading menu example shown earlier: If your test script is going to select an Audi for make, then your first wait would be for the list of menu options to contain "Audi." If you're selecting A4 for the model, then wait on "A4" to appear in the model list.

Dynamic search results lists are another common problem area, but they're solved in exactly the same way. Consider Google, which uses AJAX calls to give you a "look ahead" feel as you're typing your query in. If you're wanting your script to search for Amelia Earhart, then you might input "ameli" to start the search. The proper wait condition for this example would be for the results list to contain "ameliaearhart" at which point you know it's safe to select that item and move on with the step that selects that entry.



Work with Developers

The scenarios above are fairly simple ones. Many times multiple AJAX calls will be happening concurrently, or there may be other UI complexities falling into place. Rather than struggling with dealing with these situations, teams working on UI automation should be working as a whole team—developers pairing up closely with testers or those creating automation scripts.

Developers are able to quickly clear up page lifecycle issues for testers. Moreover, developers can help make the system's UI more testable in AJAX scenarios. It's a trivial matter for a developer to add a hidden, empty HTML element on the page after a complex set of concurrent AJAX has completed. This gives testers something simple to latch their wait statements on, eliminating confusion around what particular conditions the script is trying to delay on.

Learn the Patterns, Learn Your System

Learning to appropriately handle asynchronous situations in your systems' user interfaces is a critical part of building stable, valuable automation suites that won't kill your team with maintenance costs. Take some time to learn how the pages in your system deal with asynchronous situations, and ensure you're spending lots of time talking with the developers responsible for those pages. You'll find your frustration levels dropping and your success rates climbing.

To be continued in next issue...

Jim is the VP for ALM and Testing at Falafel Software. He has been in various corners of the IT world since joining the US Air Force in 1982. He's spent time in LAN/WAN and server management roles in addition to many years helping teams and customers deliver great systems. Jim has worked with organizations ranging from start-ups to Fortune 100 companies to improve their delivery processes and ship better value to their customers. Jim's been in many different environments but greatly prefers those adopting practices from Lean and Agile communities.

When not at work you might find Jim in the kitchen with a glass of wine, playing Xbox, hiking with his family, or banished to the garage while trying to practice his guitar.

CONTINUED FROM PAGE 12...

In my article, I suggested to find people in testing community who you aspire to, and strive to become their peer though excelling at your craft as a tester. Here is my advice: find the best in the world. Ask them for mentorship. Learn from them, and become the best in the world yourself. For the names of my peers and people I found worth aspiring to, I refer you to my article and to any CAST or Let's Test conference programs.

You have been working in testing field from quite some time. Do you think there are any advancements happening in this field?

Of course! On the outside, the new thinking in software development and delivery, like Lean/Kanban and Agile concepts, introduced new ways for people on the team to work together. These changes impact testers as well. On the inside, members of Context Driven Testing community advance the testing craft through defining and enriching the concept of a skilled tester.

What are the key ingredients to become successful Test Manager? And what are those skills you actively seek out in fresh testers when hiring?

The secret of a successful manager resides in the success of his or her team members. The key ingredients for the Test Manager's success is making the testers' jobs fulfilling and rewarding, while providing them with growth opportunities. When I hire testers into my team, I look for an "open mind" first: whether they are open to try new processes, practices and, in general, other ways of doing things. Then, I look for the attitude: whether the person is sociable, and can work closely with professionals of other disciplines: developers, business analysts, designers, etc. Last (but not least), I seek out their passion for quality.

What would be your advice for ladies who aspire to take higher roles in organization (like Senior Test Manager, Test Director etc.)? Any tips, dos and don'ts to share?

My advice for anyone, who is aspiring to take higher roles in organization is to understand the responsibilities and develop the skills which are necessary for such positions. Most of the time, you will be representing the work of others in front of management. Also, you will be making decisions which impact the people who work for you. The skills that you need to develop are good communication (upward and downward), building the environment of trust, and development of a good long-term strategy for your business/professional unit.

My advice for ladies in particular is to recognize the signs of cultural bias towards women in management positions and learn to address it. When I mention gender bias, I don't mean for women to be alarmed with it, it's being ready to deal with it, like getting an umbrella when it's raining outside. For example, there are several power dynamics techniques in public speaking. Mastering these techniques will help a woman speak in front of any audience successfully. Also, I would wholeheartedly recommend getting a mentor.

How meaningful do you find traditional ways of test reporting (Pass/Fail and number based) when it comes to feeding that information to higher management?

Feeding information to another person or department requires an understanding on how the information will be used. Usually, the higher management wants to know about the risks when they make decisions on software release/shipment. Reporting the 'number of executed test cases' or pass/fail results to management seems an unwise decision to me – as numbers don't tell the full story, and oftentimes give a false sense of security.

What is your general opinion about the ways testing gets measured in industry (be it testing progress, effectiveness, tester's performance etc.)? Is there anything we should change about it?

The question about ways testing gets measured in the industry is very broad, as the answers will vary significantly from organization to organization. For example, in my product team the 'classic' test cases are created for user acceptance testing, performed by business analysts. It may make sense for them; I would not interfere with that practice. However, my team of testers does not 'execute test cases'; my testers are involved in pair testing with development, exploratory testing of new and existing features and workflows, and maintaining a robust set of 'safety net' automation checks. In my opinion, the best measurement of testing effectiveness is qualitative, not quantitative. I think we need to make that concept more accepted by the industry at large.

You are also known as Context Driven Testing Scholar. Please tell us about your Context Driven journey and how it has affected your thought process around testing.

I identified myself as the Scholar in Context Driven Testing, because the essence of CDT school of thought is the educated learning. When I first got introduced to CDT principles, I called them "common sense". Later on, I got more and more interested in all aspects of context driven testing, and how it fits within each process methodology or within the goal of the organization. With time, my understanding of 'testing' broadened too. I found it fascinating that most of the time anyone of us is continuously putting to test: someone else's assumptions, our environment, people we work with, and any incoming information, be it what we observe, read or hear.

CDT made it easier for me to understand and communicate with other professionals in software field. There is a concept of software development as a never-ending learning process; the team doesn't know what it will build until they build it. Testing is an integral part of this discovery process.

You are conference co-chair of CAST this year. Please help us imagine that awesome picture. What is CAST going to be all about this year? Any special plans and attractions?

Imagine people from every part of the world coming to discuss the latest trends in software testing while getting inspired by their peers. Imagine comradery, exchange of ideas and lots of fun. Imagine bringing your questions and problems to the conference, and leaving with solutions, new friends and desire to excel in your craft. That's what CAST is to me every year (this will be my 5th CAST). It's a community event, very motivational for software testers. All you need is a desire to learn and share. The CAST never starts or ends 'on time' – the discussions, games, hands on testing activities are happening from early mornings to late nights. CAST is a life-changing event to any professional, who thinks seriously about being a tester.

Would you like to convey any message for our readers?

It's an opportunity for me to invite everyone to CAST 2014. The theme of this year's conference is Art and Science of Testing. It attracted the most renowned thinkers and leaders of worldwide testing community. And it's in the heart of New York, the City that never sleeps! I promise you a good time. It's a guarantee.

Last question, how has your experience with Tea-time with Testers been? We would love to get your feedback and suggestions.

I enjoy reading Tea Time with Testers, and sometimes experience a bit of information overload! Your magazine is always packed with articles written by most admired names in our industry. It's an amazing achievement. I wish you best of luck, Tea Time with Testers! Thank you for talking to me today.

Happiness is….

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

Like our FACEBOOK page for more of such happiness

https://www.facebook.com/TtimewidTesters

Call for Articles

Theme – Automation beyond Regression

Last date of submission – 25th August 2014

Contact: editor@teatimewithtesters.com

Click HERE to read our Article Submission FAQs!

To move rapidly, freeze!

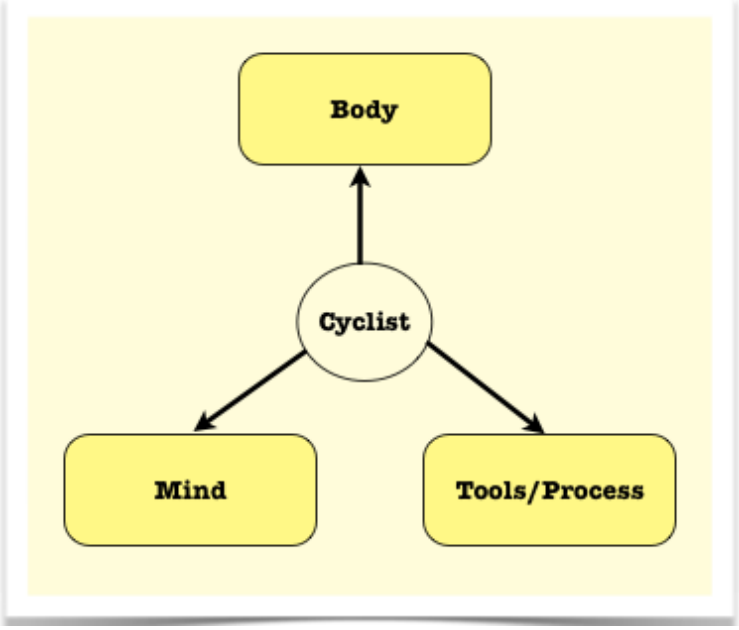
"The expanding expectations and the contracting effort/time/cost" was the title of the panel discussion in the recently concluded STeP-IN conference where I was one of the panelists. The discussion centered on the notion of the expanding needs and expectations from the customers and the business pressure of accomplishing them in shorter times with tighter budgets.

The points put forth included early involvement by QA, deepening the domain skills, getting adept at programming and aggressively tools, embracing shorter cycles of development, moving from detection("illness")to ; prevention ("wellness"). My focus was on the human aspects as how our thinking can enable us to accelerate what we do and also do significantly better.

Let me tell you a story now. A few months ago I did a self supported long cycle ride (brevet) - 1000km in about 70 hours. Riding a cycle for three days continuously in heat, winds, cold, climbs, and rain is demanding. It requires strength and fitness. And riding long hours, against unrelenting wind and demanding climbs can significantly slow you down and requires a strong mind filled with positive energy. And most importantly it requires an extreme focus on NOW, to be mindful, not be worried

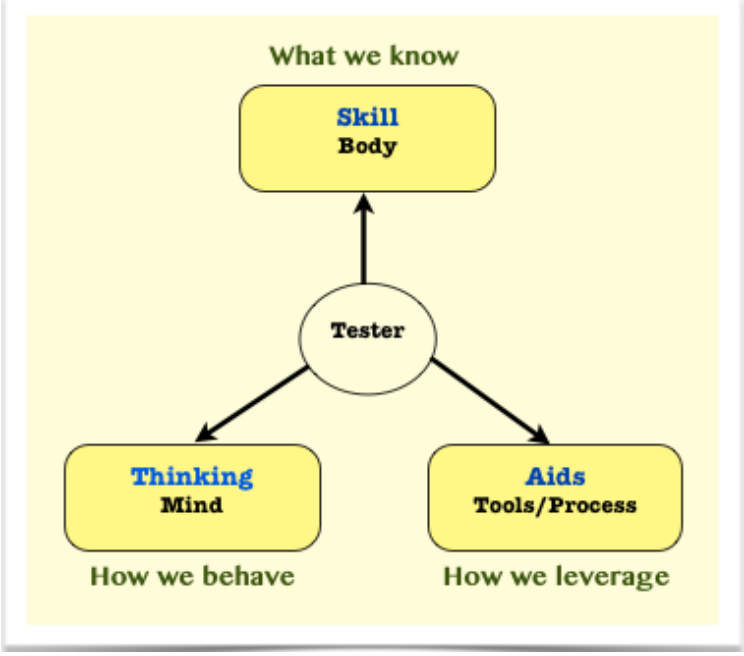
about the cutoff times in the future. Stay in the present, pedal strong and enjoy. And of course, a cycle with good technology and a power-generating yet energy conserving process of cycling.

This is illustrated in the picture alongside. As a cyclist I see the accomplishing of long distance more in less time with lower effort not as a brute force exercise, but as one that requires the power of mind to be effectively harnessed. It is about staying in the moment, living every moment, unmindful of the future and therefore generating more power to go all the way.



I discovered that when I became mindful, the power output was enormous, the legs were in rapid circular motion, no pain, just the rhythm of movement and quiet breathing. The wind stopped bothering me, the hills/climbs were inviting and the speedometer stayed constantly high. And to me this was Wow! And that is when I discovered the power of mindfulness.

Now let us relate this to the discussion on hand. When expectations increase, testing scope increases and it seems that we need to possess more skills (akin to strength of the body). It requires the tester to be adept in the domain so that he/she can get into the end user shoes. To be adept in technology/tools so that that automation can be exploited. And this relates to the "Skills - What we know". And this is indeed important.



And the natural extension is to exploit technology to test faster. To inject probes into the system that can enable better testing, to automate as much as possible, to relegate work to the machine and also things faster. And by increasing the process agility, we can to be faster and crisply focus on critical things. And this is "How we leverage" the technology/tools/process.

So we can accelerate and accomplish more, by becoming more skilled and exploiting technology/tools/process. Is that all? Nah. "How we respond and behave" has a significant contribution. It is not just about knowledge/skill and but how we use it. The ability to empathize

enables the tester to get in to the end-users mind and therefore understand the expanding expectations. An extremely positive frame of mind to cope with the demanding timeframes goes a long time in staying calm and not to buckle under pressure. And finally staying focused in the present during a test session does wonders. It enables one to be mindful and unleashes the mindful creativity and skills just flow. And this requires you to think about deadline, the future. It is just about being in the current test session. Staying in the context, absorbing it and using it well. It is about freezing the present.

A short mindful session of testing is like this equation: $P=Lt_{t->0} W/t$ where P=Outcome productivity, W=Work to be done and t=Time to do. So when we kind of freeze the time, our Productivity zooms. And this implies that we be very conscious of now, stay focused in the present and enjoy the test session. Staying mindful makes the whole job of testing far more fun and enjoyable, it is no more activity or a chore.

And this was my discussion point in the panel discussion - how the human aspects related to mind/thinking enable us to accelerate what we do and also do significantly better. Harness the power of mind in addition to skills/technology/tools/process.

Want to experiment with mindfulness? Take a glass of water, sip on it slowly, focus sing on the water and drinking only. Feel the water caressing your lips, swishing your tongue, feel the quiet taste of water and the water quietly descending though the food pipe. Spend a few minutes and you will enjoy this. Time stops. Would you not want any session related to testing be like this?

For a great future, stay in the present. To move rapidly and accomplish more, freeze!

To mindful testing and joy…

Cheers.

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

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

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

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.

Claim your Smart Tester of The Month Award. Send us your answer for Crossword b4 25th July 2014 & grab your Title.

Send -> editor@teatimewithtesters.com with Subject: Testing Crossword

1		2		3		4			5
	■		■		■		■	■	
6		7		8					
	■		■		■		■	■	
9		10		11			12		
	■		■		■	■		■	■
13							14		

Horizontal:

1. It is a test automation framework which drives off the UI of Android native and hybrid applications and the mobile web (10)

6. It is a load and performance testing tool for web applications (9)

9. Testing to determine whether the system/software meets the specified usability requirements, in short form (2)

10. The level of business importance assigned to an individual item or test (8)

13. It is a Test Execution Management System that takes care of the administration, reporting, and sequencing of the tests, providing a single common user interface for all of the tests that you develop (7)

14. Testing based on an analysis of the internal structure of the component or system, in short form (3)

Vertical:

1. It is a free cross-platform Functional Testing solution (6)

2. The process of creating demand on a system or device and measuring its response, in short form (2)

3. It is testing process where the system validated against the invalid input data, in short form (2)

4. An underlying factor that caused a non-conformance and possibly should be permanently eliminated through process improvement, the first word (4)

5. Testing which demonstrates that the system under test does not work, the first word (5)

7. A variable that is read by the component (5)

8. Checks for memory leaks or other problems that may occur with prolonged execution, in short form (2)

10. A path which cannot be exercised by any set of possible input values, in short form (2)

11. A group of people whose primary responsibility is to conduct software testing for other companies, in short form (3)


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 - A Thoughtful Eye

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 **FREE** copy ,

Subscribe to our group at

Google™

Join our community on

facebook

Follow us on

www.teatimewithtesters.com

Give Feedback