

A yellow mug filled with tea sits on a wooden desk. To its left is a spiral-bound notebook with a black pen resting on it. In the bottom right corner, a portion of a black computer keyboard is visible.

Tea-time with Testers

Your tea-time eZine on software testing

January, 2011

Year 1 Issue I

Editorial

Hello Everyone,

Hebbel has said that, "Nothing great in the world has ever been accomplished without passion."

With an honest passion for doing dedicated job in our very own testing community, we are happy to present "Tea-time with Testers" e-magazine to all of you.

This monthly ezine is an outcome of couple of my sleepless nights and number of tea-time discussions with my colleague & co-founder of this initiative Mr. Pratik Kumar Patel.

I still remember the very first day when I was put into software testing and also the confused state of my mind, me being unaware of the beauty and pleasure behind finding things out, which I would say software testing gives me today. I must admit that online software testing communities and various QA blogs were those who taught me how mighty the field of testing is & brought confidence about my field and never-fading smile on my face.



By publishing this ezine we promise to offer an open platform where we all can discuss, share, suggest, contribute, criticize, guide, relax and many other activities related to the world of software testing.

We will be more than just happy if we succeed to bring the same confidence and never-fading smile on yet another new tester's face.

I would like to thank Pratik for supporting this idea and being there with me. Special thanks to our all authors for their support and contribution.

We are sure that you all will join and support us in taking this initiative further.

Yours sincerely


Lalit kumar Bhamare



Editorial2

What's making the news?

Top List of 9 Big Programming Failures of 20105

Speaking Tester's Mind

Should Tester's abilities be based on their defect count 8

by Adam Brown

Test Framing.....11

by Michael Bolton

Teaching programmers to test.....15

by Joel Montvelisky

In the School of Testing

The SCRUM Primer21

by Pete Deemer, Gabrielle Benefield, Craig Larman, Bas Vodde

Weekend Testing and it's benefits to a fresher30

by Ajay Balamurugadas

Dice game: Key life skills for testers32

by Darren Mcmillan

What type of Tester are you ?38

by Shrini Kulkarni



T' talks

Aesthetics in Software Testing.....45

by T. Ashok

Tool Watch

SAHI50

by Chris Philip

Next Issue60

Our Team61



What's making News?

- find out the latest happenings in the technology world

Image: www.bigfoto.com

TOP LIST OF 9 BIG PROGRAMMING FAILURES OF 2010

As in most any year in the modern age, 2010 saw its share of problems due to programming errors, bugs and other shortcomings. A timing error with the New York Stock Exchange (NYSE) systems, a Skype crash, problems with Chase online banking, faulty automobile systems at Toyota and privacy breaches were among the year's big name software problems that might have been avoided at the programming level. For instance, the Skype outage, which affected a large number of Skype's 560 million users, resulted from a bug in the Windows version of the software. In a blog post, Lars Rabbe, CIO at Skype, said the company needed to enhance its capability for testing for and detecting bugs. "While our Windows v5 software release was subject to extensive internal testing and months of Beta testing with hundreds of thousands of users, we will be reviewing our testing processes to determine better ways of detecting and avoiding bugs which could affect the system," Rabbe said in his post. Software application defects are inevitable, but developers have a better chance of catching them if they use enhanced debugging and code analysis platforms and employ Agile methods. In addition, companies such as Replay Solutions, Coverity, Corensic and others offer software to help developers detect and resolve programming issues during the development lifecycle.

1. McAfee's Glitch

Security company McAfee in April issued a software update to protect computers against a list of malicious files. Unfortunately, a file that is part of Microsoft's Windows operating system made it onto the list. McAfee's software deleted this file, causing affected computers to shut down and start on a continuous reboot cycle. The glitch primarily hurt McAfee corporate customers and was widely reported and criticized. Even multibillion-dollar software companies aren't perfect, it seems.



2. Skype Crash

A bug in the Windows version of Skype's Internet telephony software caused the service to crash for nearly 24 hours in late December. Skype suffered a serious server outage Dec. 22 that left swaths of its 560 million or so worldwide users without PC-calling capabilities for most of the day. In a blog post, Skype's CIO, Lars Rabbe, blamed a bug in the Windows version for the problem.

3. Russian Satellite

In December, the world saw the successful launch of the final three GLONASS navigation satellites. However, just after launching, a programming error caused the carrier rocket to veer off course and lose the Proton-M booster rocket carrying the satellites in the Pacific Ocean north of Hawaii.

4. NYSE Timing Error

A timing error on a software update at NYSE Euronext's electronic Arca exchange prompted an exchange-traded fund tracking the Standard & Poor's 500 index to drop nearly 10 percent.

5. Android Kernel Bug

In November, an analysis of the kernel used in Google's Android smartphone software turned up 88 high-risk security flaws that could be used to expose users' personal information.

6. Facebook Privacy CSRF

Facebook, under intense scrutiny this year for its privacy policies and procedures, was discovered to have a cross-site request forgery flaw that could have allowed hackers to alter profile information and change privacy settings for individual pages.

7. Chase Online Banking

Software from a third-party database company corrupted information in Chase systems and prevented users from logging on. The service was down for two days and had an impact on millions of customers.

8. Toyota Electronic Data Recorder

A software bug in electronic data recorders for Toyota (the black box that records the speed of the automobile) was found to have created incorrect readouts. This is significant as the automaker is under scrutiny for reports of unexplained acceleration and numerous recalls this year. A report from NASA and the NHTSA [National Highway Traffic Safety Administration] is due in 2011 to determine the cause of the unexplained acceleration, but it has also been speculated to be the result of electronic issues.

9. AT&T

Heavy demand for upload capacity from the iPhone 4 exposed a flaw in the software for Alcatel-Lucent's 3G network equipment, causing users of iPhone4G to experience abnormally slow upload speeds. These issues were widely reported on by mainstream and consumer publications.

News source: [eweek](#)





Speaking Tester's Mind

- straight from the author's desk

Image: www.bigfoto.com

Should Tester's abilities be based on their defect count?



by Adam Brown

Image: renjith krishnan / FreeDigitalPhotos.net

I've been looking at one software testing service provider a fair bit recently and I toyed with the idea of signing up for a little while. From what I can gather, it seems to put a lot of emphasis on how many defects a particular tester is raising and then works out their abilities based on that. A Tester who raises more valid defects is more valuable to the community than their counterparts who raise fewer defects. Wait... What?

There are several reasons why a Tester's abilities can't be measured by looking at the number of defects they raise. In the examples below I will use Tester A and B who are both identical in terms of abilities and are working on the same piece of software at the same time. The two are equal in every single way. Imagine a set of twins who have done everything together since they were born and now happen to work at the same company.

Type of defects

Let's say that the project the two testers are working on spans 6 weeks. During that time Tester A raises 6 defects and Tester B raises 30 defects. Tester A's defects are all critical, show stopping defects and Tester B's defects are all minor cosmetic issues. Who is the better Tester in this scenario?

Setting targets

Let's say that Testers are told from the outset that their pay rises and performance reviews are based on the number of defects they raise. Of course they're going to try and get their defect count up to try and make themselves look better.



An example that comes to mind when I think about setting targets is an example that my Dad always uses. He used to work as an engineer for a large white goods firm (I can't name them for legal reasons, but I will call them "RotJoint" from this point forward) which involved him going out to customer's houses to fix problems. The Engineers had targets to meet such as "Right first time" which meant that Engineers aimed to fix a problem on the first call out. The minimum figure that RotJoint had set the Engineers was at least 60% right first time. Now there were Engineers who were getting 100% Right first time month after month, and others that were getting 45% – 50%. From a management perspective, who's the better Engineer? Clearly the Engineer who's getting 100% Right First Time is the better Engineer... right?

It turns out that a lot of those engineers who were getting 100% Right First Time weren't correcting the problem at all. They were temporarily fixing the problem, then either going back in their own time to fix it, or the temporary fix would cause a NEW problem so that the Engineer would have to be called out again.

Rob Lambert made a great point on Twitter a while back:

[@Rob Lambert](#): ... if I have to find 20 defects as part of my contract, I'll find 20 defects.

I've taken this out of context a little and this was part of a much larger conversation, but the point is still valid. If your target is to find 20 defects, you **will** find 20 defects.

Type of Testing performed

The type of Testing be carried out also has to be taken in to consideration. The number of defects found during Regression Testing is very low, and sometimes even zero. Tester's A and B are working on separate projects now. Both are roughly the same size, but Tester A's is a change to an existing application and Tester B's is a completely new application. Whilst Tester A is performing Regression Testing, the likelihood of them finding defects is quite low, although they may raise one or two. Tester B on the other hand has raised 20 as they are already well in to their Integration and System Testing by now. If this happened to coincide with performance reviews, Tester A may appear to be the less competent Tester.

Size of projects

The size of the project will also have an effect on the defect count. Larger projects will tend to have proportionately more defects. But this is obviously because there is more to test!

Complexity of Software

The type and complexity of the Software under Test can also have an impact on the number of defects raised. If the Software is quite complex in nature then the likelihood is that there is more chance of something going wrong and the developer making a mistake. Whereas Software that's quite simple and basic may yield fewer defects because it was quite straight forward to code.

So in conclusion

Would I like my skills to be assessed on how many defects I raised on a project? Absolutely not. I don't think it's fair that some companies would use this as a measure of tester's abilities. I get the impression that companies that use this metric don't really understand a great deal about Software Testing and just want *something* to put in a pretty graph. The logic is that Testers raise defects, so who ever raises the most is better than someone else, but that's like saying "My mechanic is rubbish! It's broken down twice this year where as my wife, who goes to a different mechanic, never breaks down!"



Biography



Adam Brown has 5 years of working experience in the IT industry, with four years working as a Software Tester and one year as a Developer.

He has completed his education from Trinity 6th Form in Information Technology, Business Studies, Resistant Materials, and Physics.

He has both developed and tested Warehouse Distribution and Stock Management systems, and tested Retails System POS, Mail Order, E-Commerce, Online Booking, Held Terminals, Kiosks as well as Clinical Trials and drug supply management systems.

His specialties are Blogging, Social Media, Retail and Warehouse Management Systems, Mail Ordering Systems.

He has sported the variety of roles in different organizations like IT Director at Square Sprocket Ltd, QA Test Analyst at Medoc Computers Ltd and C Programmer at Optimise Systems Ltd.

Adam can be reached on his blog at

<http://testing.gobanana.co.uk/>

And also on twitter @brownie490

Our beginners and aspiring
Tester friends,

As promised, we have
something in store for you.

Keep reading to know
more...!

- Editor





Test Framing

by Michael Bolton

Image: www.bigfoto.com

Test framing is the set of logical connections that structure and inform a test. To test is to compose, edit, narrate, and justify two stories. One is a story about the product--what it does, how it does it, how it works, and how it might not work--in ways that matter to your clients. The other is a story about your testing--how you came to know and understand the product story. The testing story comprises several crucial elements--how you designed your tests, how you configured, operated, observed and evaluated the product, what you haven't tested yet or won't test at all, why what you did was good enough, and what what you haven't done isn't so important. Of course, the story must be a true account of the testing work. To build the tests and the story expertly requires a skill that we call test framing.

Over several years of training and consulting, I have observed that many testers need help in one or more aspects of test framing--designing tests, evaluating the results, telling the testing story, or making the connection between the testing mission and the test performed, in an unbroken chain of narration, logic, and justification of cost versus value.

The basic idea is this: in any given testing situation

- You have a testing mission (a search for information and your mission may change over time).
- You have information about requirements (some of that information is explicit, some implicit; and it will likely change over time).
- You have risks that inform the mission (and awareness of those risks will change over time).



- You have ideas about what would provide value in the product, and what would threaten it (and you'll refine those ideas as you go).
- You have a context in which you're working (and that context will change over time).
- You have oracles that will allow you to recognize a problem (and you'll discover other oracles as you go).
- You have models of the product that you intend to cover (and you'll extend those models as you go).
- You have test techniques that you may apply (and choices about which ones you use, and how you apply them).
- You have lab procedures that you follow (that you may wish to follow more strictly, or relax).
- You have skills and heuristics that you may apply.
- You have issues related to the cost versus the value of your activities that you must assess.
- You have time (which may be severely limited) in which to perform your tests.
- You have tests that you (may) perform (out of an infinite selection of possible tests that you could perform).
- You configure the product (typically in a variety of ways).
- You operate the product (also in a variety of ways).
- You observe the product (directly, or mediated by tools; observing both behaviour and state).
- You evaluate the product (in particular by applying the heuristic oracles noted above).

Test framing involves the capacity to follow and express a direct line of logic that connects the mission to the tests. Along the way, the line of logical reasoning will typically touch on elements between the top and the bottom of the list above. The goal of framing the test is to be able to answer questions like

- Why are you running (did you run, will you run) this test (and not some other test)?
- Why are you running that test now (did you run that test then, will you run that test later)?
- Why are you testing (did you test, will you test) for this requirement, rather than that requirement?
- How are you testing (did you test, well you test) for this requirement?
- How does the configuration you used in your tests relate to the real-world configuration of the product?
- How does your test result relate to your test design?
- Was the mission related to risk? How does this test relate to that risk?
- How does this test relate to other tests you might have chosen?



- Are you qualified (were you qualified, can you become qualified) to test this?
- Why do you think that is (was, would be) a problem?

The form of the framing is a line of propositions and logical connectives that relate the test to the mission. A proposition is a statement that expresses a concept that can be true or false. We could think of these as affirmative declarations or assumptions. Connectives are word or phrases ("and", "not", "if", "therefore", "and so", "unless", "because", "since", "on the other hand", "but maybe", and so forth) that link or relate propositions to each other, generating new propositions by inference. This is not a strictly formal system, but one that is heuristically and reasonably well structured.

Here's a fairly straightforward example:

GIVEN: (The Mission :) Find problems that might threaten the value of the product, such as program misbehaviour or data loss.

Proposition: There's an input field here.

Proposition: Upon the user pressing Enter, the input field sends data to a buffer.

Proposition: Unconstrained input may overflow a buffer.

Proposition: Buffers that overflow clobber data or program code.

Proposition: Clobbered data can result in data loss.

Proposition: Clobbered program code can result in observable misbehaviour.

Connecting the propositions: IF this input field is unconstrained, AND IF it consequently overflows a buffer, THEREFORE there's a risk of data loss OR program misbehaviour.

Proposition: The larger the data set that is sent to this input field, the greater the chance of clobbering program code or data.

Connection: THEREFORE, the larger the data set, the better chance of triggering an observable problem.

Connection: IF I put an extremely long string into this field, I'll be more likely to observe the problem.

Conclusion: (Test :) THEREFORE I will try to paste an extremely long string in this input field AND look for signs of mischief such as garbage in records that I observed as intact before, or memory leaks, or crashes, or other odd behaviour.

Now, to some, this might sound quite straightforward and, well, logical. However, in our experience, some testers have surprising difficulty with tracing the path from mission down to the test, or from the test back up to mission—or with expressing the line of reasoning immediately and cogently.

Our approach, so far, is to give testers something to test and a mission. We might ask them to describe a test that they might choose to run; and to have them describe their reasoning. As an alternative, we might ask them why they chose to run a particular test, and to explain that choice in terms of tracing a logical path back to the mission.



If you have an unframed test, try framing it. You should be able to do that for most of your tests, but if you can't frame a given test right away, it might be okay. Why? Because as we test, we not only apply information; we also reveal it. Therefore, we think it's usually a good idea to alternate between focusing and defocusing approaches. After you've been testing very systematically using well-framed tests, mix in some tests that you can't immediately or completely justify. One of the possible justifications for an unframed test is that we're always dealing with hidden frames. Revealing hidden or unknown frames is a motivation behind randomized high-volume automated tests, or stress tests, or galumphing, or any other test that might (but not certainly) reveal a startling result. The fact that you're startled provides a reason, in retrospect, to have performed the test. So, you might justify unframed tests in terms of plausible outcomes or surprises, rather than known theories of error. You might encounter a "predictable" problem, or one more surprising to you. In

To test is to tell two parallel stories: a story of the product, and the story of our testing. James and I believe that test framing is a key skill that helps us to compose, edit, narrate, and justify the story of our testing in a logical, coherent, and rapid way. Expect to hear more about test framing, and please join us (or, if you like, argue with us) as we develop the idea.

Biography



Michael Bolton has over 20 years of experience in the computer industry testing, developing, managing, and writing about software & has been teaching software testing and presenting at conferences around the world for nine years.

He is the co-author (with senior author James Bach) of Rapid Software Testing, a course that presents a methodology and mindset for testing software expertly in uncertain conditions and under extreme time pressure.

Michael can be reached through his Web site, <http://www.developsense.com>





Teaching programmers to test

by Joel Monvelisky

Image: www.bigfoto.com

Would you trust a programmer to test your application? It's like asking a Fox to guard the chicken-house, right?

Well, sometimes you don't have a choice, or you do but that choice is to release the application untested...

As part of a consulting engagement I am doing at a friend's company, yesterday I started providing short training sessions for programmers who need to learn how to test better. It's not that this company doesn't have testers, they have good testers! But as many other agile teams they have much more testing tasks than available testers, and they want programmers to take part in at least some of their testing tasks.

Programmers are not good at testing!

A couple of months ago I wrote a post explaining why I think programmers make poor testers, and I still think that in a sense it is like asking a dog to fly (take a look at the cool picture in the post). On the other hand I also believe that with some good intentions, hard work, and perseverance you can teach a dog to skip far enough and that way to jump over large obstacles.



In the same way, you will not be able to magically transform programmers into a good tester overnight, but you can start by working with them on their weaknesses and then teaching them some simple and effective testing techniques. With good intentions, a little practice, some hand-holding, and constant feedback you will be able to get some good testing help in your project.

Step 1 – understand the limitation and weaknesses of a programmer

I started my session yesterday by going over the points I listed in my earlier blog that make programmers “less-than-perfect-testers”:

- Parental feeling towards their own code.
- Placing the focus mainly on positive scenarios, instead of actively looking for the bugs.
- Tendency to look at a “complex problem” as a collection of “smaller, simpler, and isolated cases”.
- Less End-to-End or User-Perspective oriented.
- Less experience and knowledge of the common bugs and application pitfalls.

It made a big difference during the session to not only list the weaknesses but talk about why each of them is naturally present in programmers due to the nature of their work, their knowledge and training.

Step 2 – how to plan tests

I’ve seen that many (most?) programmers have the perception that testing requires little or no planning.

Maybe we testers are guilty of this misconception since we don’t get them involved in our test planning process as much as we should, but the truth of the matter is that when we ask them to test they they automatically grab a mouse and start pressing on the buttons without much consideration or thought.

Test Planning is a cardinal aspect of good tested, so I gave them a couple of ideas and principles planing:

1. DON’T TEST YOUR OWN CODE!

When as a team they are asked to test, they should make sure to divide the tasks so that each of them is testing the code developed by other programmers as much as possible.

2. Work with your testing team to create test sets.

This specific team is using PractiTest (the hosted QA and Test Management platform my company is developing), but it can be any other tool or format (even word and excel!). They should sit with their testers and define what test cases need to be run, reusing the testing scripts already available in their repositories.

3. Expand your scenarios by making “Testing Lists”

- Features that were created or modified (directly or indirectly)
- User profiles and scenarios to be verified.
- Different environments / configurations / datasets to be tested



The use of these lists is two-fold.

They help you get a better idea of what you want to test while you are running your manual test scripts, and they also serve as a verification list to consult towards the end of the tests when you are looking for additional ideas or when you want to make sure you are not missing anything of importance.

4. Testing Heuristics – SFDEPOT (read San Francisco Depot)

The use of (good) heuristics greatly improves the quality of your testing.

I provided the programmers with the heuristic I learned first and still helps me up to this day. I read about SFDEPOT from James Bach some years ago – you can check one of the sources for it from Jame's Site.

SFDEPOT stands for:

Structure (what the product is)

Function (what the product does)

Data (what it processes)

Platform (what it depends upon)

Operations (how will it be used)

Time (when will it be used)

There are other heuristics and Mnemonics you can take from the Internet...

Step 3 – what to do when running tests

We talked a lot about tips to help perform good testing sessions.

1. Have a notebook handy.

In it you can take notes such as testing ideas you will want to test later, bugs you ran into and want to report later in order to "not to cut your testing thoughts", etc.

2. Work with extreme data.

Big files vs. Small files

Equivalent input classes [-10 ; 0 ; 1 ; 10,000,000 ; 0.5 ; not-a-number]

Dates: Yesterday, now, 10 years from now etc.

3. Think about negative scenarios.

- How would Mr. Bean use your software?
- How would a hacker try to exploit the system?
- What would happen if...? (blackout, run out of space, exceptions, etc)
- What if your 2 year old would hijack the keyboard in the middle of an operation? etc.



4. Focus & Defocus.

I used this technique to explain to them that during their testing process they need to make a conscious effort to always look at the bigger picture (application, system, process) and not only focus on the specific function they were testing.

5. Fight Inattentional Blindness.

I used the following film of the kids passing the balls to explain the concept of Inattentional Blindness and it worked great!

We were 8 people in the room, and only I had seen the video before. Out of the 7 participants one is currently a tester, another one is a former tester turned programmer, the rest are "regular programmers".

The cool thing is that only the 2 testers saw the Gorilla the first time... talk about making a point!

Step 4 – what to do when (you think) you are done testing

We talked about how even when you think you are done testing you should always make sure there is nothing else that should be tested, and what techniques they can use in order to find these additional places:

1. Walking the dog

Taking a break from your tests, doing another task for 30 – 60 minutes, and then returning to review the tests you did and what you found. This break usually helps to refresh the mind and to come up with more ideas.

2. Doing a walk-through session to review the tests you did.

Most of the times you will be able to get more ideas as you explain to your peers about tests you just did.

The funny part is that many of these ideas will come from yourself and the things you think about when you are explaining your tests to others out loud.

3. Ask for ideas from other developers or testers.

Simple as it sounds, come to others in your team and ask them to pitch you ideas of stuff they think you could test. 90% of the stuff you will already have tested, but the other 10% might prove useful too!

In the end is a question of mindset and motivation

One of the things I like most of agile teams (and many non-agile but yes smart development teams) is that they define Quality to be the responsibility of the whole team and not only of the testing guys running the tests tasks at the end of the process.

My final piece of advice to the group was that everything starts from them, and their understanding that testing is not a trivial task and definitely not a reprimand for doing a bad job or for finishing their tasks ahead of time. What's more, I am sure that once these guys start testing better and gaining a testing perspective of their application they will also start developing better software too.



Biography



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

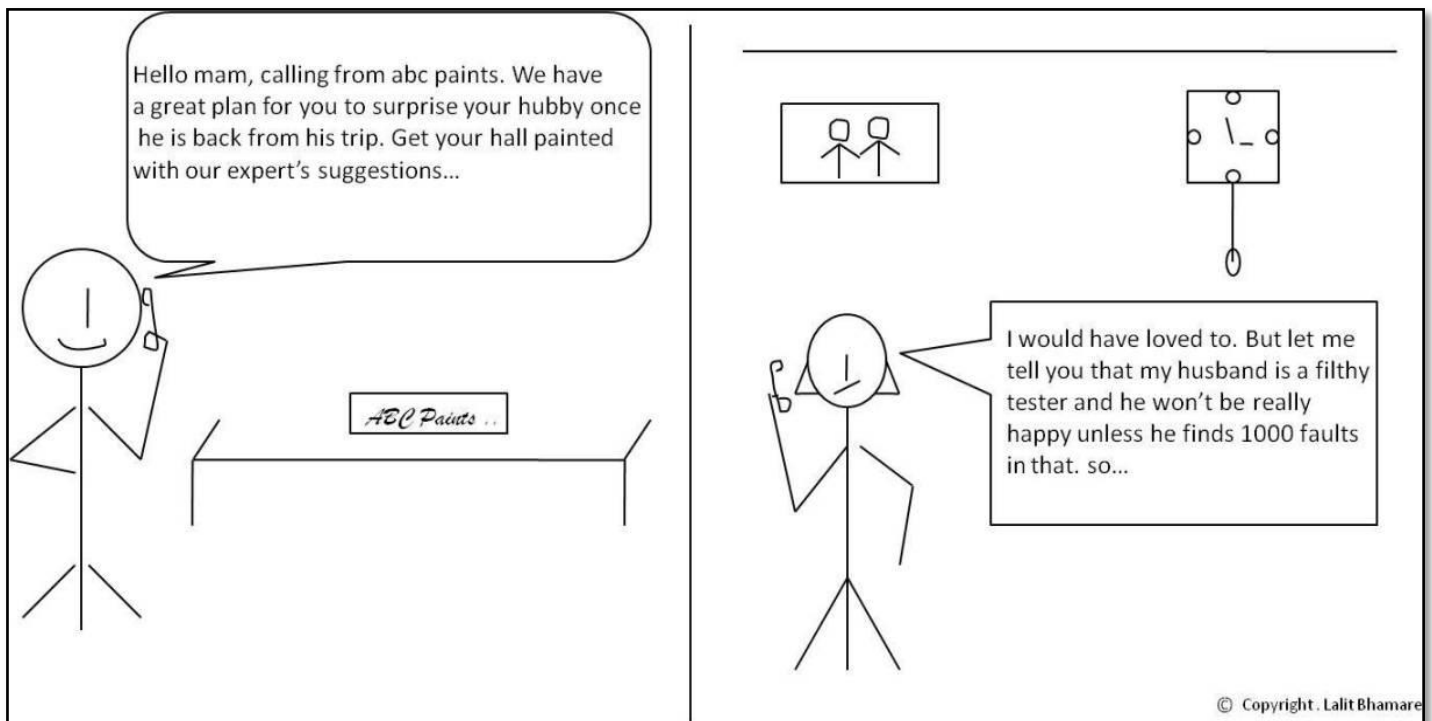
He's worked in companies ranging from small Internet Start-Ups and all the way to large multinational corporations, including Mercury Interactive (currently HP Software) where he managed the QA for TestDirector/Quality Center, QTP, WinRunner, and additional products in the Testing Area.

Today Joel is the Solution and Methodology Architect at PractiTest, a new Lightweight Enterprise Test Management Platform.

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

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

Tickle your Testing Bone... ☺



some schemes are not meant for a tester's family

Inspiration: Mr. Andy Glover. The [cartoon-tester](#). Thanks for showing us something new about craft of testing, Andy - Editor



In the School of Testing

for better learning & sharing experience



Image: nuttakit / FreeDigitalPhotos.net





Traditional Software Development

The traditional way to build software, used by companies big and small, was a sequential life cycle commonly known as “the waterfall.” There are many variants (such as the V-Model), but it typically begins with a detailed planning phase, where the end product is carefully thought through, designed, and documented in great detail. The tasks necessary to execute the design are determined, and the work is organized using tools such as Gantt charts and applications such as Microsoft Project. The team arrives at an estimate of how long the development will take by adding up detailed estimates of the individual steps involved. Once stakeholders have thoroughly reviewed the plan and provided their approvals, the team starts to work. Team members complete their specialized portion of the work, and then hand it off to others in production-line fashion. Once the work is complete, it is delivered to a testing organization (some call this Quality Assurance), which completes testing prior to the product reaching the customer. Throughout the process, strict controls are placed on deviations from the plan to



ensure that what is produced is actually what was designed.

This approach has strengths and weaknesses. Its great strength is that it is supremely logical – think before you build, write it all down, follow a plan, and keep everything as organized as possible. It has just one great weakness: humans are involved.

For example, this approach requires that the good ideas all come at the beginning of the release cycle, where they can be incorporated into the plan. But as we all know, good ideas appear throughout the process – in the beginning, the middle, and sometimes even the day before launch, and a process that does not permit change will stifle this innovation. With the waterfall, a great idea late in the release cycle is not a gift, it's a threat.

The waterfall approach also places a great emphasis on writing things down as a primary method for communicating critical information. The very reasonable assumption is that if I can write down on paper as much as possible of what's in my head, it will more reliably make it into the head of everyone else on the team; plus, if it's on paper, there is tangible proof that I've done my job. The reality, though, is that most of the time these highly detailed 50-page requirements documents just do not get read. When they do get read, the misunderstandings are often compounded. A written document is an incomplete picture of my ideas; when you read it, you create another abstraction, which is now two steps away from what I think I meant to say at that time. It is no surprise that serious misunderstandings occur.

Something else that happens when you have humans involved is the hands-on “aha” moment – the first time that you actually use the working product. You immediately think of 20 ways you could have made it better. Unfortunately, these very valuable insights often come at the end of the release cycle, when changes are most difficult and disruptive – in other words, when doing the right thing is most expensive, at least when using a traditional method.

Humans are not able to predict the future. For example, your competition makes an announcement that was not expected. Unanticipated technical problems crop up that force a change in direction. Furthermore, people are particularly bad at planning uncertain things far into the future – guessing today how you will be spending your week eight months from now is something of a fantasy. It has been the downfall of many a carefully constructed Gantt chart.



In addition, a sequential life cycle tends to foster an adversarial relationship between the people that are handing work off from one to the next. "He's asking me to build something that's not in the specification." "She's changing her mind." "I can't be held responsible for something I don't control." And this gets us to another observation about sequential development – it is not much fun. The waterfall model is a cause of great misery for the people who build products. The resulting products fall well short of expressing the creativity, skill, and passion of their creators. People are not robots, and a process that requires them to act like robots results in unhappiness.

A rigid, change-resistant process produces mediocre products. Customers may get what they first ask for (at least two translation steps removed), but is it what they really want once they see the product? By gathering all the requirements up front and having them set in stone, the product is condemned to be only as good as the initial idea, instead of being the best once people have learned or discovered new things.

Many practitioners of a sequential life cycle experience these shortcomings again and again. But, it seems so supremely logical that the common reaction is to turn inward: "If only we did it better, it would work" – if we just planned more, documented more, resisted change more, everything would work smoothly. Unfortunately, many teams find just the opposite: the harder they try, the worse it gets! There are also management teams that have invested their reputation – and many resources – in a waterfall model; changing to a fundamentally different model is an apparent admission of having made a mistake. And Scrum is fundamentally different...

Agile Development and Scrum

The agile family of development methods were born out of a belief that an approach more grounded in human reality – and the product development reality of learning, innovation, and change – would yield better results. Agile principles emphasize building working software that people can get hands on quickly, versus spending a lot of time writing specifications up front. Agile development focuses on cross-functional teams empowered to make decisions, versus big hierarchies and compartmentalization by function. And it focuses on rapid iteration, with



continuous customer input along the way. Often when people learn about agile development or Scrum, there's a glimmer of recognition – it sounds a lot like back in the start-up days, when we “just did it.”

By far the most popular agile method is Scrum. It was strongly influenced by a 1986 Harvard Business Review article on the practices associated with successful product development groups; in this paper the term “Rugby” was introduced, which later morphed into “Scrum” in Wicked Problems, Righteous Solutions (1991, DeGrace and Stahl) relating successful development to the game of Rugby in which a self-organizing team moves together down the field of product development. It was then formalized in 1993 by Ken Schwaber and Dr. Jeff Sutherland. Scrum is now used by companies large and small, including Yahoo!, Microsoft, Google, Lockheed Martin, Motorola, SAP, Cisco, GE, CapitalOne and the US Federal Reserve. Many teams using Scrum report significant improvements, and in some cases complete transformations, in both productivity and morale. For product developers – many of whom have been burned by the “management fad of the month club” – this is significant. Scrum is simple and powerful.

Scrum Summary

Scrum is an iterative, incremental framework for projects and product or application development. It structures development in cycles of work called Sprints. These iterations are no more than one month each, and take place one after the other without pause. The Sprints are timeboxed – they end on a specific date whether the work has been completed or not, and are never extended. At the beginning of each Sprint, a cross-functional team selects items (customer requirements) from a prioritized list. The team commits to complete the items by the end of the Sprint. During the Sprint, the chosen items do not change. Every day the team gathers briefly to inspect its progress, and adjust the next steps needed to complete the work remaining. At the end of the Sprint, the team reviews the Sprint with stakeholders, and demonstrates what it has built. People obtain feedback that can be incorporated in the next Sprint. Scrum emphasizes working product at the end of the Sprint that is really “done”; in the case of software, this means code that is integrated, fully tested and potentially shippable. Key roles, artifacts, and events are summarized in Figure 1.



A major theme in Scrum is “inspect and adapt.” Since development inevitably involves learning, innovation, and surprises, Scrum emphasizes taking a short step of development, inspecting both the resulting product and the efficacy of current practices, and then adapting the product goals and process practices. Repeat forever.

SCRUM

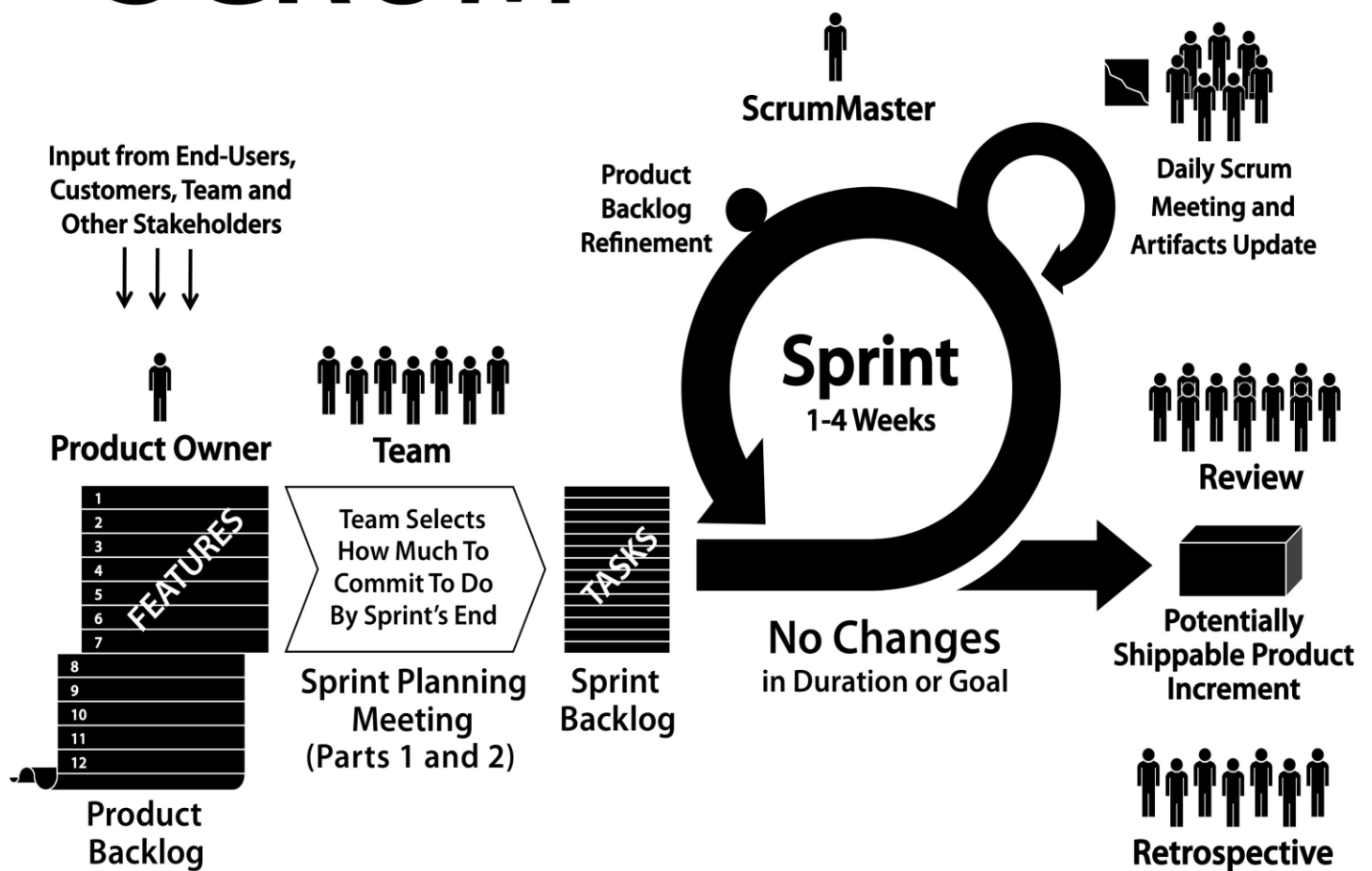


Figure 1. Scrum



Scrum Roles

In Scrum, there are three roles: The Product Owner, The Team, and The ScrumMaster.

Together these are known as The Scrum Team. The Product Owner is responsible for maximizing return on investment (ROI) by identifying product features, translating these into a prioritized list, deciding which should be at the top of the list for the next Sprint, and continually re-prioritizing and refining the list. The Product Owner has profit and loss responsibility for the product, assuming it is a commercial product. In the case of an internal application, the Product Owner is not responsible for ROI in the sense of a commercial product (that will generate revenue), but they are still responsible for maximizing ROI in the sense of choosing – each Sprint – the highest-business-value lowest-cost items. In practice, ‘value’ is a fuzzy term and prioritization may be influenced by the desire to satisfy key customers, alignment with strategic objectives, attacking risks, improving, and other factors. In some cases, the Product Owner and the customer are the same person; this is common for internal applications. In others, the customer might be millions of people with a variety of needs, in which case the Product Owner role is similar to the Product Manager or Product Marketing Manager position in many product organizations. However, the Product Owner is somewhat different than a traditional Product Manager because they actively and frequently interact with the Team, personally offering the priorities and reviewing the results each two- or four-week iteration, rather than delegating development decisions to a project manager. It is important to note that in Scrum there is one and only one person who serves as – and has the final authority of – Product Owner, and he or she is responsible for the value of the work. The Team builds the product that the Product Owner indicates: the application or website, for example. The Team in Scrum is “cross-functional” – it includes all the expertise necessary to deliver the potentially shippable product each Sprint – and it is “self-organizing” (self managing), with a very high degree of autonomy and accountability. The Team decides what to commit to, and how best to accomplish that commitment; in Scrum lore, the Team is known as “Pigs” and everyone else in the organization are “Chickens” (which comes from a joke about a pig and a chicken deciding to open a restaurant called “Ham and Eggs,” and the pig



having second thoughts because “he would be truly committed, but the chicken would only be involved”).

The Team in Scrum is seven plus or minus two people, and for a software product the Team might include people with skills in analysis, development, testing, interface design, database design, architecture, documentation, and so on. The Team develops the product and provides ideas to the Product Owner about how to make the product great. In Scrum the Teams are most productive and effective if all members are 100 percent dedicated to the work for one product during the Sprint; avoid multitasking across multiple products or projects. Stable teams are associated with higher productivity, so avoid changing Team members. Application groups with many people are organized into multiple Scrum Teams, each focused on different features for the product, with close coordination of their efforts. Since one team often does all the work (planning, analysis, programming, and testing) for a complete customer-centric feature, Teams are also known as feature teams.

The ScrumMaster helps the product group learn and apply Scrum to achieve business value. The ScrumMaster does whatever is in their power to help the Team and Product Owner be successful. The ScrumMaster is not the manager of the Team or a project manager; instead, the ScrumMaster serves the Team, protects them from outside interference, and educates and guides the Product Owner and the Team in the skillful use of Scrum. The ScrumMaster makes sure everyone (including the Product Owner, and those in management) understands and follows the practices of Scrum, and they help lead the organization through the often difficult change required to achieve success with agile development. Since Scrum makes visible many impediments and threats to the Team’s and Product Owner’s effectiveness, it is important to have an engaged ScrumMaster working energetically to help resolve those issues, or the Team or Product Owner will find it difficult to succeed. There should be a dedicated full-time ScrumMaster, although a smaller Team might have a team member play this role (carrying a lighter load of regular work when they do so). Great ScrumMasters can come from any background or discipline: Engineering, Design, Testing, Product Management, Project Management, or Quality Management.



The ScrumMaster and the Product Owner cannot be the same individual; at times, the ScrumMaster may be called upon to push back on the Product Owner (for example, if they try to introduce new deliverables in the middle of a Sprint). And unlike a project manager, the ScrumMaster does not tell people what to do or assign tasks – they facilitate the process, supporting the Team as it organizes and manages itself. If the ScrumMaster was previously in a position managing the Team, they will need to significantly change their mindset and style of interaction for the Team to be successful with Scrum.

Note there is no role of project manager in Scrum. This is because none is needed; the traditional responsibilities of a project manager have been divided up and reassigned among the three Scrum roles. Sometimes an (ex-)project manager can step into the role of ScrumMaster, but this has a mixed record of success – there is a fundamental difference between the two roles, both in day-to-day responsibilities and in the mindset required to be successful. A good way to understand thoroughly the role of the ScrumMaster, and start to develop the core skills needed for success, is the Scrum Alliance's Certified ScrumMaster training. In addition to these three roles, there are other contributors to the success of the product, including functional managers (for example, an engineering manager). While their role changes in Scrum, they remain valuable. For example:

- They support the Team by respecting the rules and spirit of Scrum
- They help remove impediments that the Team and Product Owner identify
- They make their expertise and experience available

In Scrum, these individuals replace the time they previously spent playing the role of “nanny” (assigning tasks, getting status reports, and other forms of micromanagement) with time as “guru” and “servant” of the Team (mentoring, coaching, helping remove obstacles, helping problem-solve, providing creative input, and guiding the skills development of Team members). In this shift, managers may need to change their management style; for example, using Socratic questioning to help the Team discover the solution to a problem, rather than simply deciding a solution and assigning it to the Team... **(to be continued in the next issue)**





Biography



Pete Deemer is a founder of GoodAgile, and co-founder of the Scrum Training Institute, which is headed by Dr. Jeff Sutherland, the co-creator of Scrum. Pete is based in Singapore and specializes in helping companies in India, China, and Australia succeed with Scrum. He has provided Scrum training and coaching to some of the largest companies in the world, including services companies such as Infosys, Wipro, TCS, and Cognizant and product companies such as Nokia, Ericsson, GE, and EMC. Pete is the co-author of The Scrum Primer, one of the most widely read introductions to Scrum, and lead author of The Distributed Scrum Primer, a guide to distributed and multilocation Scrum.

Pete is an honors graduate of Harvard University, and has spent the last 22 years leading teams building products and services at global companies. Most recently he served as Vice President of Product Development for Yahoo!, where he led Yahoo's global adoption of Scrum, which grew to over 2000 developers worldwide during his tenure. He spent a number of years as adjunct faculty at University of California Berkeley, where he received the prestigious Club 6 teaching award. Pete is a visiting lecturer at the Institute of Systems Science at the National University of Singapore for the 2010-2011 academic year.

Pete can be reached at his mail id –
petedeemer@scrumtraininginstitute.com

Gabrielle Benefield is a Certified Scrum Trainer based in London offering classes in the United Kingdom and Europe. Gabrielle has over 18 years experience building enterprise software and web products at global companies and is a founder of the Scrum Training Institute, with Jeff Sutherland, the co-creator of Scrum, Pete Deemer and Jens Ostergaard.

Gabrielle works with clients from diverse industries including banking, telecommunications, and internet. Gabrielle was Senior Director of Agile Development at Yahoo! leading Yahoo!'s large-scale corporate adoption of Scrum, which encompassed more than 200 teams projects and over 1500 employees in the US, India, Europe, and Asia. She is co-author of The Scrum Primer (www.scrumprimer.com), the most widely read introductory guide to Scrum available and became the 12th Certified Scrum Trainer in the world, being personally certified by Ken Schwaber. Gabrielle provides training and coaching in many Agile and Lean methodologies.



Craig Larman works as the lead coach of lean product development adoption at Xerox, and serves as a consultant for large-scale Scrum and enterprise agile adoption at Nokia and Siemens Networks (now, NSN), at Statoil and Kongsberg Maritim and Cisco-Tandberg (in Norway), at Alcatel-Lucent, and at Schlumberger and UBS, among many other clients. Craig has served as chief scientist at Valtech, a consulting, outsourcing, and skills transfer organization with divisions in many countries, with a division in Bangalore that applies agile methods to offshore development. In his role at Valtech, he created "agile offshore development" while living in India and China. His work focuses on product groups that involve a few hundred to a few thousand people, usually multisite.

He is one of the earliest Practicing ScrumMasters and one of the first worldwide authorized to coach and certify new ScrumMasters and Product Owners, as a Certified Scrum Trainer.

Craig holds a B.Sc. and M.Sc. in computer science from beautiful SFU in Vancouver, BC, with research emphasis in artificial intelligence (having little of his own).

To know more about Craig please visit -
http://www.craiglarman.com/wiki/index.php?title=Craig_Larman

He can be reached at his mail id - craig@craiglarman.com

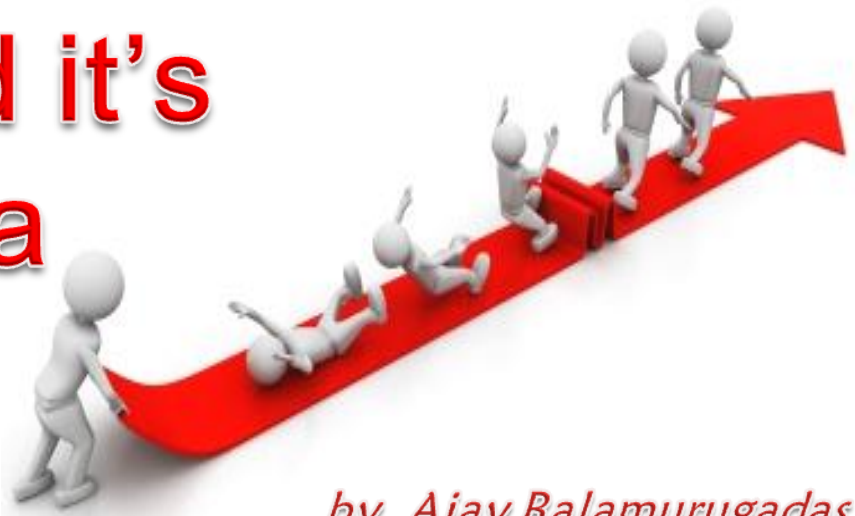


Bas Vodde is originally from Holland, however he has lived in China, Finland, China again and currently lives in Singapore. He led the Agile transformation project at Nokia Networks and later Nokia Siemens Networks, and currently works for his own company called Odd-e. He's been working with several large products and several large company change projects.

He's still also developing software himself and is one of the authors of the C++ unit test framework CppUTest.

Bas can be reached at his website www.odd-e.com

Weekend Testing and it's benefits to a fresher



by Ajay Balamurugadas

Image: renjith krishnan / FreeDigitalPhotos.net

Q: I am a fresher. I want to improve my testing skills. How will Weekend Testing help me?

A: Welcome to Weekend Testing (WT). I am happy for you for two reasons:
You are aware of WT & WT is one of the good platforms to improve your knowledge.

Let me highlight some of the ways WT can benefit you:

1. An excellent platform to test your skills.

WT has this brilliant idea of coming with a unique mission every session. Though there are chapters in India, Europe, Australia-New Zealand, Americas and European WeekNight Testing as of today, every chapter makes sure that no mission-product combination is repeated. Every session is a totally different experience and what's cool is you are free to participate in any of the sessions - No chapter restrictions.

2. Varied experience

If you have no experience of working in any organization, I cannot think of any other easy way to face such a diverse set of products & missions. Just imagine, after every weekend you would have tested 2-3 products and at least one would interest you to explore more. Last week, I learnt about [Rapid Reporter](#), [Text2MindMap](#) and [Process Monitor](#). Worst case, you might learn about 1 tool every weekend. Isn't that cool? Different tools, different experience added to your ammunition list.



3. Excellent repository of experience reports

If you are not satisfied by the sessions and want more, the WT website -www.weekendtesting.com must each report, chat transcript at your own pace and ask us questions. We are ready to help you. You might have an idea which no one of us thought of. We are waiting to hear that idea.

WT is all about Test, Learn and Contribute.

4. Cost Vs Value

We ask for two hours of your time and nothing else. Two hours of pure learning, interactions with other testers, new products, new friends, new test ideas. Think about the value you get for just two hours of investment. Cost Vs Value. WT is aware that you are dedicating your valuable time and hence tries its best to give you a learning platform. Make use of it.

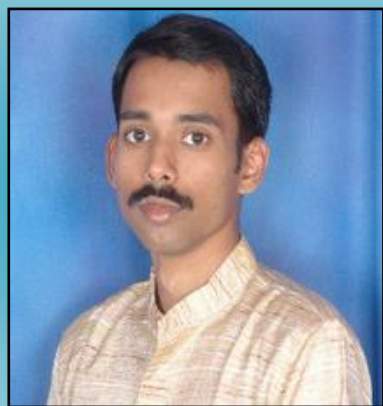
5. Be part of the community

Join us on twitter, create your own blog, add us on Gmail, Skype. Lets talk. So many WT sessions are discussed even after the sessions on different forums, blogs and on twitter. Take part in those discussions. You might learn something new. Build your reputation, demonstrate your testing skills and be part of the testing community. Your next organization's QA lead might be someone whom you paired with in a weekendtesting session!

Looking forward to your participation in weekendtesting sessions.

P.S. Readers are encouraged to read about Weekend testing - introduction [here](#)

Biography



Ajay Balamurugadas is a passionate software tester ready to learn to test any software. He has been awarded scholarship by Software Testing Club and is a brown-belt student of Miagi-Do school run by Matt Heusser.

Ajay shares his learning & thoughts on <http://EnjoyTesting.blogspot.com> and tweets @ajay184f



Dice game: Key life skills for testers

by Darren Mcmillan

Back in November 2010 I had the chance to attend Michael Bolton's excellent Rapid Software Testing course in London. Obviously for me this was a fantastic opportunity & experience. From those three days with Michael I had one thing that really stood out. Although not for everyone, the dice game that Michael & his colleague James Bach tour the world playing with students, for me really emphasises some key essential skills for any software tester. I enjoyed it so much that as soon as I'd returned I quickly set out to purchase the props required to play it with my colleagues.

The Rapid Software Testing challenge

So it all started at the end of our Rapid Software Course. Thursday the 4th of November; I remember it well as Michael had been enjoying playing the role of a problematic stakeholder, who'd just wanted stuff done for the past couple of days. This challenge wasn't any different; he still gave very little away!

So Michael explained the rules of the dice challenge & laid four different coloured sets of dice on the table. Each colour set contained five dice, so twenty in total. The rules were simple! Roll the dice. In response to this, Michael would say a number. Then you'd have to reliably predict the number that Michael would say, and you've won the game. Yeah that's real simple I thought; can't be that hard right?

We were split into four groups; each group had three to four people in it. We begun discussing amongst ourselves our plans & all very quickly had begun rolling dice and making guesses as to what the pattern could be. Michael progressed around the room responding with a number of his own, which sometimes matched our guesses and sometimes didn't. Good fun & very frustrating at the same time!



For myself I found it extremely difficult working with two people who I didn't know very well. We were all very strong characters & each of us wanted to try our own approach & tactics. We'd started out a bit of a mess! I began to note down responses in accordance with the factors that I was observing. Then things began to improve.

After some time we believed we had cracked the pattern and proudly announced to Michael that we had solved this puzzle. Chuffed we were!

Michael said something along the lines of "Ah! ,Very good!". Smug we were! Then he ran a test that tested us. He rolled the dice, and we weren't able to guess the right number after all. Before we knew it the problem had been solved by another team! How did I feel about this? Well I was pretty gutted to be honest! I like to win things like most people & to fail; well, that kind of sucked!

Michael then went on to explain the pattern & congratulated the smug bunch who'd cracked his puzzle. Fair play to them though; they'd solved the pattern; or at least one of them did I forget who. I remember Peter Houghton being part of that team; a man who is now also a fellow blogger. Perhaps it was Pete I honestly can't recall.

After a debrief from Michael on the challenge, it was a quick chance to say goodbye; shed some tears & hug each other like all real men do. Well not quite, but we did get to say our goodbyes

Trying it with colleagues

So I'd returned home & quickly set out to buy the required props. As soon as they'd arrived my fellow Glasgow based test team colleagues Andrew, Chris, Michael & Stuart all got the chance to become what would be known as "Victims" of the dice game! They were joined by around another five or six developers, a graphics designer, a couple of team leads & a documenter over the next few days.

We'd just finished our Friday conference call with our team members in Jakarta, Indonesia. I'd pulled out some dice from my bag & grouped them into colours. Five dice for each of the four colours, twenty in total. Each person received their dice & I begun to explain the rules of the challenge to them.

You can gather they were a bit skeptical by now ☺. Firstly my rules got them a bit riled up & secondly the challenge was pretty famous. Most had heard about it & had a small inkling of fear, gradually increasing with each previous day that I'd harped on about wanting to pose this challenge upon them.

So we progressed & I think it took them about forty minutes to eventually solve it. In fact, they only managed to solve it once our boss Michael had left the room, as he was effectively leading them astray by over complicating things, despite my best efforts to tell him this! "You're over complicating things!" Baffled look; followed by another attempt. "You're still overcomplicating things!". A very smart guy who through a series of events in the challenge (patterns found) began to doubt that any other valid pattern existed. This other pattern was of course the one that I was looking for.

Michael, Andrew & Stuart if I recall correctly all identify one or two very valid patterns. I was pleased for them, really! However as I'd explained to them "Yes you're correct, that's a very valid pattern! *Long Pause* However it's just not the pattern I'm looking for. Which is indeed a very valid pattern, like your Over the next few days that phrase became very common. My poor colleague Gerry who sits next to me had probably heard this at least fifty times. He chuckled inside himself a few times having been one of the first dice game "victims". Those chuckles progressed to laughter, followed by collaborative laughter. In



fact, near the end we had people hoarding round the desk laughing & seeing if people would get it! By this time Gerry's laughter had turned to tears of distress!

Our graphics designer at the time took several attempts at this & couldn't quite manage to crack it. If he actually sat down and kept trying he'd probably have solved it pretty quickly. The video below is Gerry trying to coach him into solve this challenge.

http://www.youtube.com/watch?v=_QeiQ52ufuQ&feature=player_embedded

So a bunch of people attempted this with me & pretty much everyone enjoyed it. A few people left feeling it was a waste of time. Some had really enjoyed it & quickly spread the word. Some people got it! When I say got it, I mean they actually saw the benefit of doing this challenge & could see how aspects of this would directly apply to their day to day job as developers, testers & so on.

Lessons Learnt

Myself, I'd realised these lessons the first time I'd participated in this challenge. Sitting on the other end of the table & watching how others tackled this challenge, was for me insightful to say the least.

Learning from others approach to testing

I've talked in the past about learning from the approach that others take when approaching a testing challenge. When I say challenge, this could be anything from an actual testing task such as developing a test plan, communicating problems with stakeholders or actually testing a piece of functionality. It could also be a challenge internally with other members of your team, such as this dice game or some of the other challenges myself, or others have did in the past (The usability challenge, Escape, the impossible challenge, testing the future, the Friday challenge).

I honestly believe, and from my experience have learned that my most valuable testing insights have come just from watching, analysing or investigating how others have approached a challenge. I've also learned a lot just from thinking about how I've approached a challenged myself. A prime example which I later went on to share here on my site was the keyword based testing technique I'd used to tackle my colleague Stuart challenge "Testing the future". I'd highly recommend you take a step back sometimes & look at how you or others have approached a challenge to see what you can learn from it.

Tackling the challenge

So let's summarise what people did to tackle the famous dice game challenge:

- **Simplifying problems**
 - Only one or two people actually started out simple.
 - Most quickly lost track & made it difficult for themselves.
 - Some managed to regain focus.
 - Most people over complicated things.
 - Some people kept trying the same things, often!
 - Eventually most people realised that it helped to simply their problems.
- **Focus / Defocus heuristic**
 - Some people took the instructions very seriously. Others questioned and challenged the constraints.



- One person focused / defocused that much that he'd cracked the challenge in only three minutes.
 - The next closest did it in ten!
 - This doesn't make either of them special in any way! More on that later.
- One person explained their approach was actually to use focus / defocus heuristics to solve this problem. Just like he solves his coding problems.
- **Assumptions**
 - Yes people made lots of these
 - Very few people questioned their assumptions initially
 - Most people had questioned the ones that they were aware of in the end
 - The people that did question them tended to crack the pattern quicker.

So you can quickly see three common trends here that helped people solve the challenge quicker; simplifying problems, focus & defocus heuristics & assumptions.

Simplifying Problems

Those that started simple quickly developed a good mental testing model of the problem. Obviously this is a key skill for any tester to be able to take a complex problem and break it down. By simplify problems we not only allow our model to quickly develop, we also gain much quicker coverage & as such negate risk.

Anyone can test! Being aware of key skills to make you an effective & efficient tester will set you aside from the rest.

Obviously simplifying problems isn't always valid. Why? Well examples could be that simple has already been done by others? Perhaps simple doesn't produce any results. Perhaps the project is that complex that simple in itself will take too long to work out & or achieve. That's where risk comes into play. It always pays to be aware of the key risks when testing something.

I remember a previous project that was testing initially by simplifying the problem then progressing to more complex tests. A good initial strategy! Now one plus one equals two right? Well whoever had tested this piece of functionality (I honestly can't recall who it was) had obviously started simple. Now a common test when simplifying a problem is to take one thing & add another right? So they did this and it quickly passed. However they'd only simplified in a context that they were aware of. Another context could have been users of the feature. If they'd simplified this and done their one plus one equals two they'd have found it failed terribly!

So not only is it good to be aware of the context of something it's also good to be aware that everything is fallible. Simplifying, risk, context, focus / defocus & assumptions are all fallible. As long as we are always aware of that we can attempt to counter possible fallacies. Like defects we'll never catch everything, we will though become better testers from improving our awareness of unknown unknowns & of course known unknowns. Much like we would be aware that from simplifying a problem, focusing & defocusing & making assumptions, all these can have negative effects along with their positives.



Focus / Defocus heuristics

This I find is extremely challenging & very rewarding when I can achieve it successfully.

Some examples I find help me focus / defocus are:

- Context revealing questions
- Switching my thread of work & returning later
- Simply taking a break
- Thinking how others would solve this

There are probably a few more worth mentioning, I just can't think of them just now.

Context revealing questions is a good one & one I can provide a good example from. In my role as a tester I'm often required to, or asked to provide feedback on the initial scope of a project be that one I'm responsible for or not. Often by the time I've joined we have a list of requirements in the form of simple user stories. Very minimal, not in depth & will often be fleshed out into more stories and/or sub stories of those stories at a later date.

I use a simple mnemonic to test these requirements "WWWWWH/KE". Which is simply who, what, when, where, why & how combined with my knowledge and experience.

- Who is this for?
- What is this for?
- When & by whom is it be done?
- Where is it being done? (Location code/team)
- Why is it being done? (What problem does it solve?)
- How is it being achieved?
- What questions does my knowledge of this or related products/systems produce?
- What questions does my experience of this or other related products/systems produce?

Obviously knowledge & experience are closely related & our experience might drive more passionate feedback due to direct interactions. All of these obviously have one thing in common. They provoke thoughts and lead off into sub questions of their own. This allows me to quickly focus and defocus on what's important & provide useful feedback rapidly.

This likewise could fail; we know that because everything is fallible. Why? Well, we could become too absorbed by over analysing requirements & overwhelm our recipients with the amount of feedback we've given. The feedback itself could also provide little value. The time taken to gather that feedback, could breach key deadlines.

Assumptions

I'll not go into assumptions as I'm pretty sure we all know the negatives of these. In fact I've talked about them a lot in the past. I've talked about my experience in making an assumption & missing expectations of the actual challenge when our team was presented with what I'd deemed the impossible challenge. It's worthwhile mentioning that although we should try our best to be aware of assumptions that we make. We will never be aware of all, or even most of the assumptions we make. More importantly being aware & actually doing something about them makes a huge difference.



We can actively question our assumptions as we make them. It is also worth asking yourself which of these assumption that I am aware of are worth noting? It might help identify risk, provoke discussion or even make itself into some formal documentation such as a features test plan.

Does time actually matter?

You can certainly see that there is indeed a lot to be learned from a simple game of dice! We can also see that cracking the challenge doesn't actually matter, or even doing it in the quickest time. Some people get hung up on this factor! I know myself I wanted to be the first to crack it as I felt it was important. Even James Bach himself have commented on how students who've quickly solved this challenge will do well for themselves.

It's what we take from it that counts & how we apply those aspects to our daily job as testers, developers, managers or whatever. Simplifying problems , focus / defocus & attempting to be aware of our assumptions are key life skills for anyone.

So that's it! This was a fun exciting little challenge that had inspired me to learn more about myself & others. It inspired me to become a better tester, like all good challenges do!

Biography



Darren McMillan has been working in the testing field for just over three years now. Having only become aware of the vibrant online testing community in the past year, he is already making big impressions via his web viral blog bettertesting.co.uk

With a genuine passion for all things testing he actively seeks to solve the problems others tend to accept. Having quickly made waves in his workplace, he is now seeking to share his experiences with others, in the hope that he can help people near & a far. He strongly believes that opportunities are there to be taken & actively promotes self learning to others.

When he is not testing or writing about his experiences, he enjoys nothing more than some quite family time. A proud father to a beautiful daughter he hopes that from leading by example he'll encourage her to follow her dreams.

Darren twits his thoughts [@darren mcmillan](https://twitter.com/darren_mcmillan)





I was assisting homework of my 7 yr old daughter in her science assignment. The assignment was to collect pictures of living and non-living things and make a collage. I thought about how one could approach this work and have fun while doing it. Being a tester, it is hard not to see connection in a work like this. Here is what I thought....

Let us say two testers "A" and "B" are given this homework assignment of creating collage of living and non living things.

Tester A: Living and non living things? Just collect few pictures of animals, plants, humans - paste them as living things and collect pictures of cars, buildings, mountains, bridges and paste them as non-living things. Home work done. Test pass and on to next thing...

Tester B: While thinking about traditional ways of looking at living and non living things, this tester also questions his own understanding and meaning of "living" and "non living". He looks around, reads, discusses with colleagues about these things. He, then ends up with a broader list and accompanying narration.

What is the difference between these ways of approaching a task at hand? Tester A is more focused on "completing" task with minimal mental engagement about the settings of the task. He takes "widely accepted" meanings and understanding of the things and optimizes the details of the task. Tester B, starts with questioning the premise of the task, meaning and understanding of the things required to complete the task. He seeks a broader understanding of task elements and



attempts to question biases, prejudices of his own and others around. Though on the face of it, it might appear that Tester B will take longer time to complete the task (homework), through practice, he might be able to finish the task nearly at the same time while understanding of broader aspects of the task and clarifying biases and assumptions.

Today's testing industry promotes type A testing mentality. Type A testers eventually will become "commodity" (with price tag 20 Dollars per hour – can execute 30 test cases per hour etc). I am not sure that is a good thing or not. If you are a manager or someone who manages testing as business – you would probably have (or would like to have) more of Type A testers as such people are plenty. Simply, take truck load of freshers from college, run them through 15 days of training on testing vocabulary, make them memorize some domain stuff (banking, insurance, telecom etc) , teach QTP – there you go an army of (Type A) testers ready for deployment (hence billing).

If you are a tester looking at long term fulfilling career – you would want to take path B – practice to do things differently, challenging status-quo, thinking broadly about the problem and its context etc. To become "Type B" tester you need practice – practice of doing testing (weekend testing is an excellent platform or training ground). I am afraid there is no shortcut here – no 15 day testing courses to get there.

There is a catch here... Type B testers are mostly rebels and free thinkers and often want break status quo, create new paths, explore new territories. Much of our industry is not very nice with such folks. Very few can tolerate "ever questioning", skeptical and rebellious tester in the team. In a control freak, military kind of set up where key focus of the job is to "follow instructions" or "getting job done" – Type B testers get frustrated. Some get converted into Type A or leave the organization. Sad situation but true in many cases. I hope that one day either rebel testers will learn to live harmoniously in a hostile, confirmatory environment or industry recognizes the value of these "problematic" folks and make space for them.

"Testers are like lighthouses at sea shores, headlights of an automobile" says [James Bach](#) (paraphrase). Good testers, apart from completing the task given to them, engage in parallel about "meta" things for better and deeper learning. That is a good thing (can I say for testers not for folks who manage testing as a business?).

Did someone say "Type C" tester?



Biography



Shrini Kulkarni is currently working as Test Solution Architect at Barclays Technology Center India. He has been a Test consultant (over 13+ Years of experience) specializing in Enterprise level software Testing, Test Automation and Performance Testing.

As experienced Tester, Test Manager and Testing coach/consultant Shrini can help organizations in assessing their testing, automation and performance testing practices and develop skills.

As a popular blogger on Software Testing, Shrini's writing has attracted many software testing professionals around the world.

Shrini has worked in companies like iGATE Global Solutions, Microsoft Hyderabad, Aditi Technologies Bangalore, i2 technologies Bangalore. Shrini worked in many roles covering entire spectrum of IT – like Software developer, Project lead, SQA lead, Test Lead, Test Manager.

His Specialties are:

Test Automation, Test Management, Enterprise Test Consulting, Performance Testing.

Shrini writes his software testing thoughts on his blog:

<http://shrini.blogspot.com/>

He can be reached at his mail id at shrini@gmail.com



And we are about to
announce it...

Look on to the next
page..!



"Tea-time with Testers" team is happy to announce the only one of its kind contest for our tester friends who have recently joined this fantastic world of testing or who aspire to be a good Tester !!!

Announcing... **"BUG-BOSS Challenge"** to let you win **"Smart-Tester of the Month"** and **"Smart-Blogger of the Month"** awards.

1. **"Smart-Tester of the Month"** award:-

In Brief:

Every month, we will be publishing the basic questions/assignments on software testing on our [homepage](#). Participants will have to mail us the solutions within the timelines mentioned.

Photograph and the details of the Winner of the contest will be published in successive editions of **"Tea-time with Testers"** as a 'Smart-Tester of the Month' tag holder.

So what are you waiting for?? Read the assignment [here](#) and be the first one to claim your **"Smart-Tester of the Month"** award.

2. "Smart-Blogger of the Month" award:-

In brief:

Our passionate tester friends who think they have it in them what it takes to write a page on any topic related to software testing, can reach out to us.

The write-up selected by our judges will be published in the successive editions of "Tea-time with Testers" with winner's details as well as it will be published on our blogsite - <http://tea-timewithtesters.blogspot.com/>

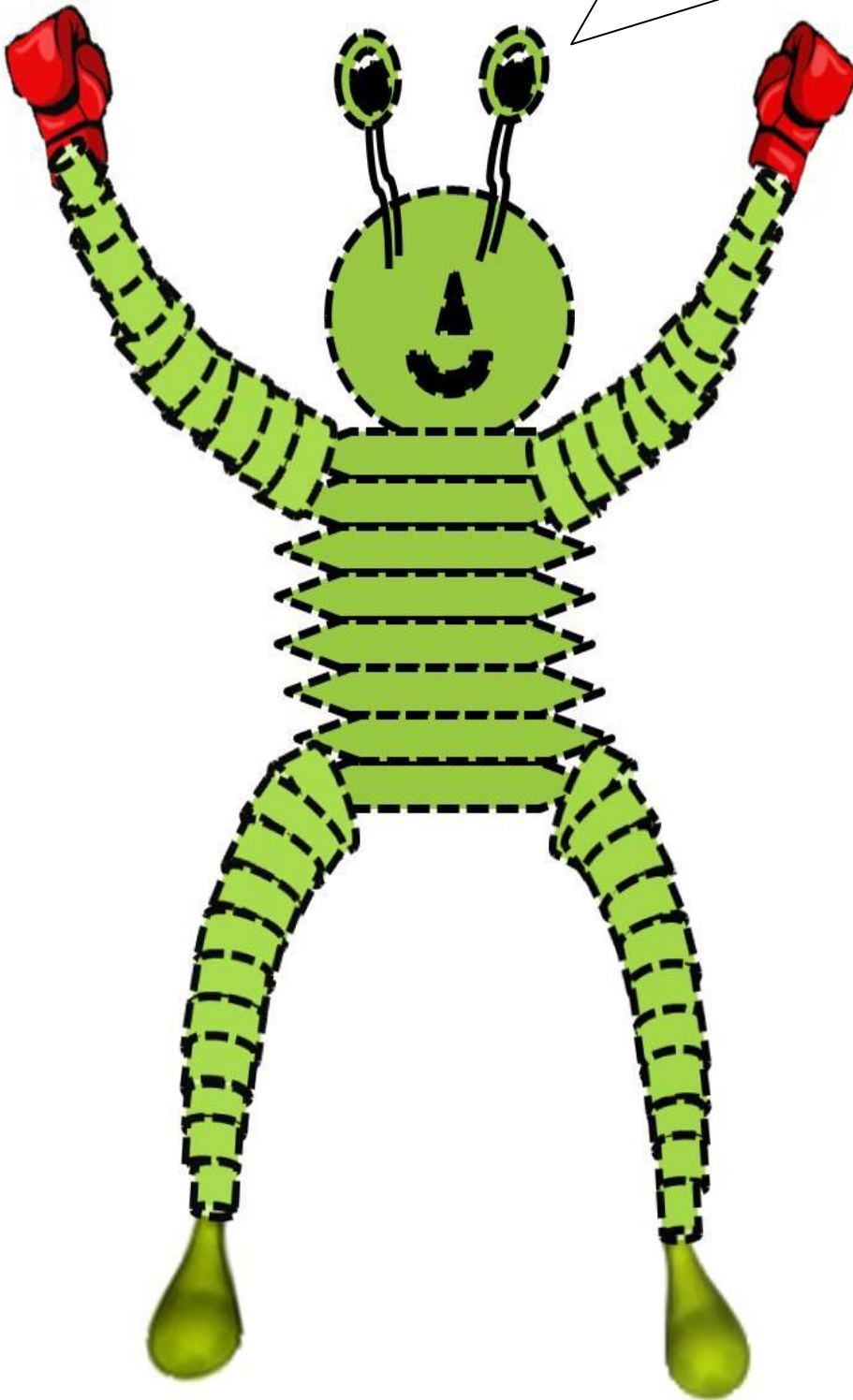
Then gear up friends. Simply mail us your article/blog-writeup at teatimewithtesters@gmail.com mentioning "Smart-Blogger of the Month" in subject.

This initiative from "Tea time with Testers" is an honest attempt to encourage our new tester friends and to promote the software testing community which has brought us till here.

We request our other friends who are already a step ahead to help us with their ideas/suggestions and by joining this venture .

- Editor

**Hee ha ha...!! so who's going to defeat
BUG-BOSS... by becoming "Smart-
Tester" & "Smart-Blogger" of the
month?**



T ' talks

software testing exclusively by T. Ashok





Aesthetics in Software Testing

Software testing is typically seen as yet another job to be done in the software development lifecycle. It is typically seen as a clichéd activity consisting of planning, design/update of test cases, scripting and execution. Is there an element of beauty in software testing? Can we see outputs of this activity as works of art?

Any activity that we do can be seen from the viewpoints of science, engineering and art. An engineering activity typically produces utilitarian artifacts, whereas an activity done with passion and creativity produces works of art, this goes beyond the utility value. It takes a craftsman to produce objects-de-art, while he takes a good engineer to produce objects with high utility value.

An object of beauty satisfies the five senses (sight, hearing, touch, smell and taste) and touches the heart whereas an object of utility satisfies the rational mind. So what are the elements of software testing that touch our heart?

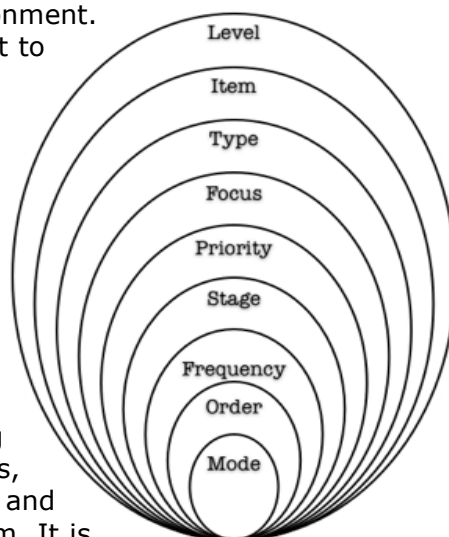
The typical view of test cases is one of utility– the ability to uncover defects; can we make test cases look beautiful? To me the form and the structure of test cases is the element of beauty in the test cases. If the test cases were organized by Quality levels, sub-ordered by items (features/modules..) then segregated by types of test, with test cases ranked by



importance/priority, and then sub-divided into conformance(+) and robustness(-), then classified by early (smoke)/late-stage evaluation, then tagged by evaluation frequency, linked by optimal execution order and finally classified by execution mode (manual/automated), we get a beautiful form and structure that not only does the job well (utility) but appeals to the sense of sight via a beautiful visualization of test cases. This is the architecture of test cases suggested by Hypothesis Based Testing (HBT).

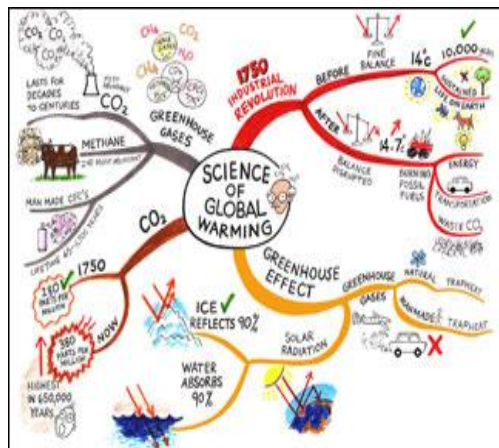
One of the major prerequisites and for effective testing is the understanding of the product and the end user's expectations. Viewed from a typical utility perspective, this typically translates into understanding of various features and intended attributes. To me the aesthetics of understanding is the ability to visualize the software in terms of the internal structure, its environment and the way end users use the software. It is about ultimately distilling the complexity into a simple singularity—to get the WOW moment where suddenly everything becomes very clear. It is about building a clear and simple map of the various types of users, the corresponding use cases and technical features, usage profile, the underlying architecture and behavior flows, the myriad internal connections and the nuances of the deployment environment. It is about building a beautiful mental mind map of the element to be tested.

Typically testing is seen as stimulating the software externally and making inferences of correctness from the observations. Are there possibly beautiful ways to assess correctness? Is it possible to instrument probes that will self assess the correctness? Can we create observation points that allow us to take better into the system? Viewing the act of evaluation from the aesthetic viewpoint, can possibly result in more creative ways to assess the correctness of behavior.



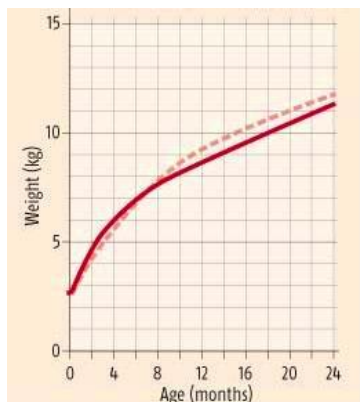
Is there aesthetics in the team structure/composition? Viewing the team collection of interesting people – specialists, architects, problem solvers, sloggers, firefighters, sticklers to discipline and geeks etc. allows us to see the beauty in the power of the team. It is not just about a team to get the job done, it is about the “RUSH” that we get about the structure that makes us feel ‘gung-ho’, ‘can-do anything’.

As professionals, we collect various metrics to aid in rational decision-making. This can indeed be a fairly mundane activity. What is aesthetics in this? If we can get extreme clarity on the aspects that want to observe and this allows us to make good decisions quickly, then I think this is beautiful. This involves two aspects—what we collect and how we present these. Creative visualization metaphors can make the presentation of the aspects of quality beautiful.



Look at the two pictures below, both of them represent the growth of a baby.





The one on the left shows the growth of a baby using the dreary engineering graph, whereas the one on the right shows the growing baby over time. Can we similarly show the growth of our baby (the software) using creative visualization metaphors?

We generate various test artifacts - test plan, test cases, reports etc. What would make reading of these a pleasure? Aesthetics here relates to the layout/organization, formatting, grammar, spelling, clarity, terseness. These aesthetic aspects are probably expected by the consumers of these artifacts today.

The test process is the most clinical and the boring aspect. Beauty is the last thing that comes to mind with respect to process. The aesthetic aspects as I see here is about being disciplined and creative, being detailed yet nimble. To me it is about devising a process that flexes, evolves in complete harmony with external natural environment. It is hard to describe these in words, it can only be seen in the mind's eye!

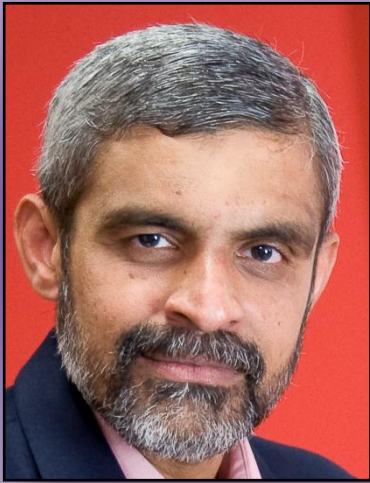
Finally on the aspect of test tooling, it is about the beautiful code that we produce to test other code. The beauty here is in the simplicity of the code, ease of understanding, modifiability, architecture and cute workarounds to overcome tools/technology limitations.

Last but not the least, aesthetics in test data is about having meaningful and real-life data sets rather than gibberish.

Beauty they say, lies in the eyes of the beholder. It takes a penchant for craftsmanship driven by passion, to not just do a job, but to produce object-de-art that appeals to the senses. As in any other discipline, this is very personal. As a community, let us go beyond the utilitarian aspects of our job and produce beautiful things. Have a great day!



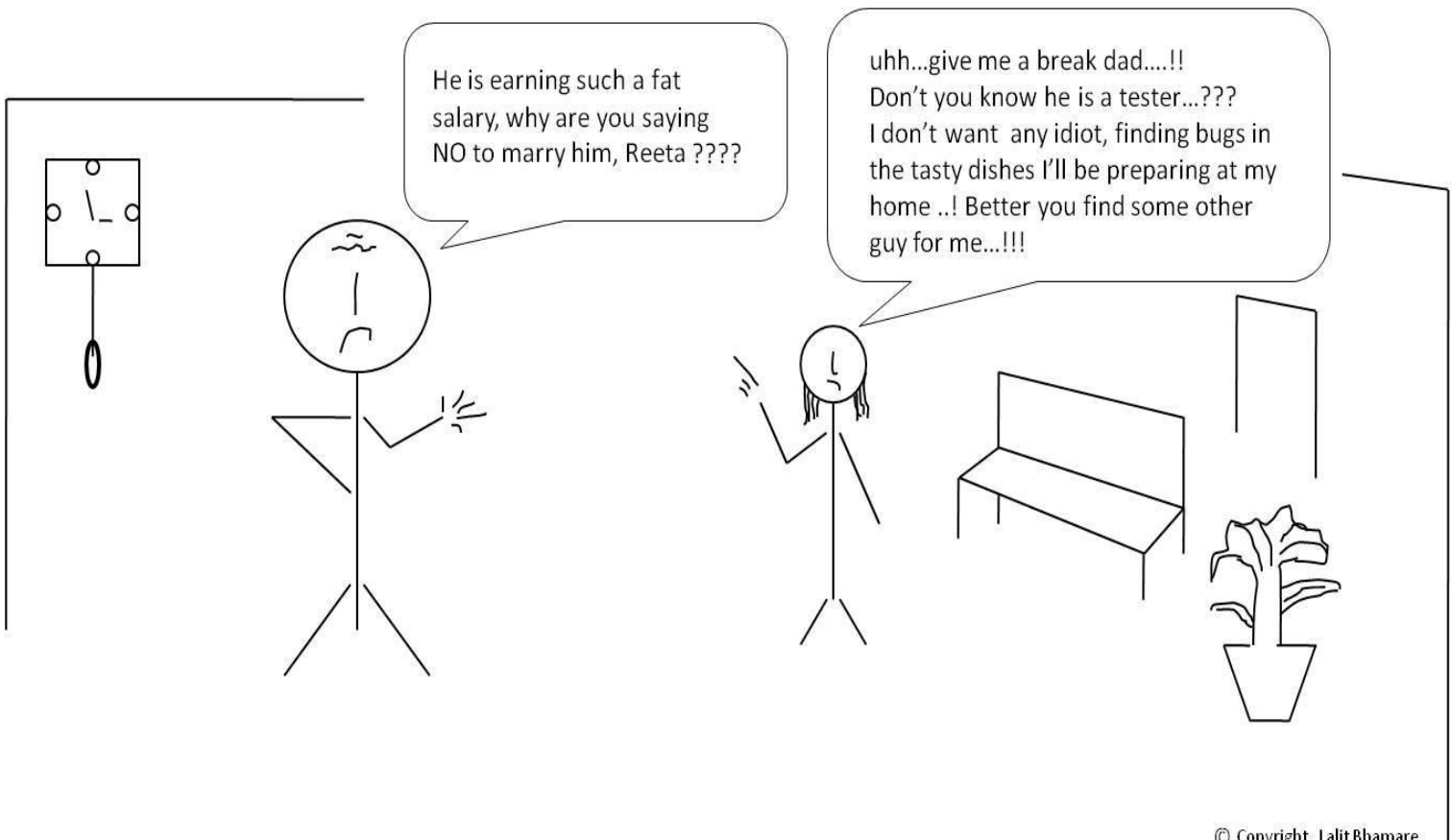
Biography



T Ashok is the Founder & CEO of STAG Software Private Limited. Passionate about excellence, his mission is to invent technologies to deliver “clean software”.

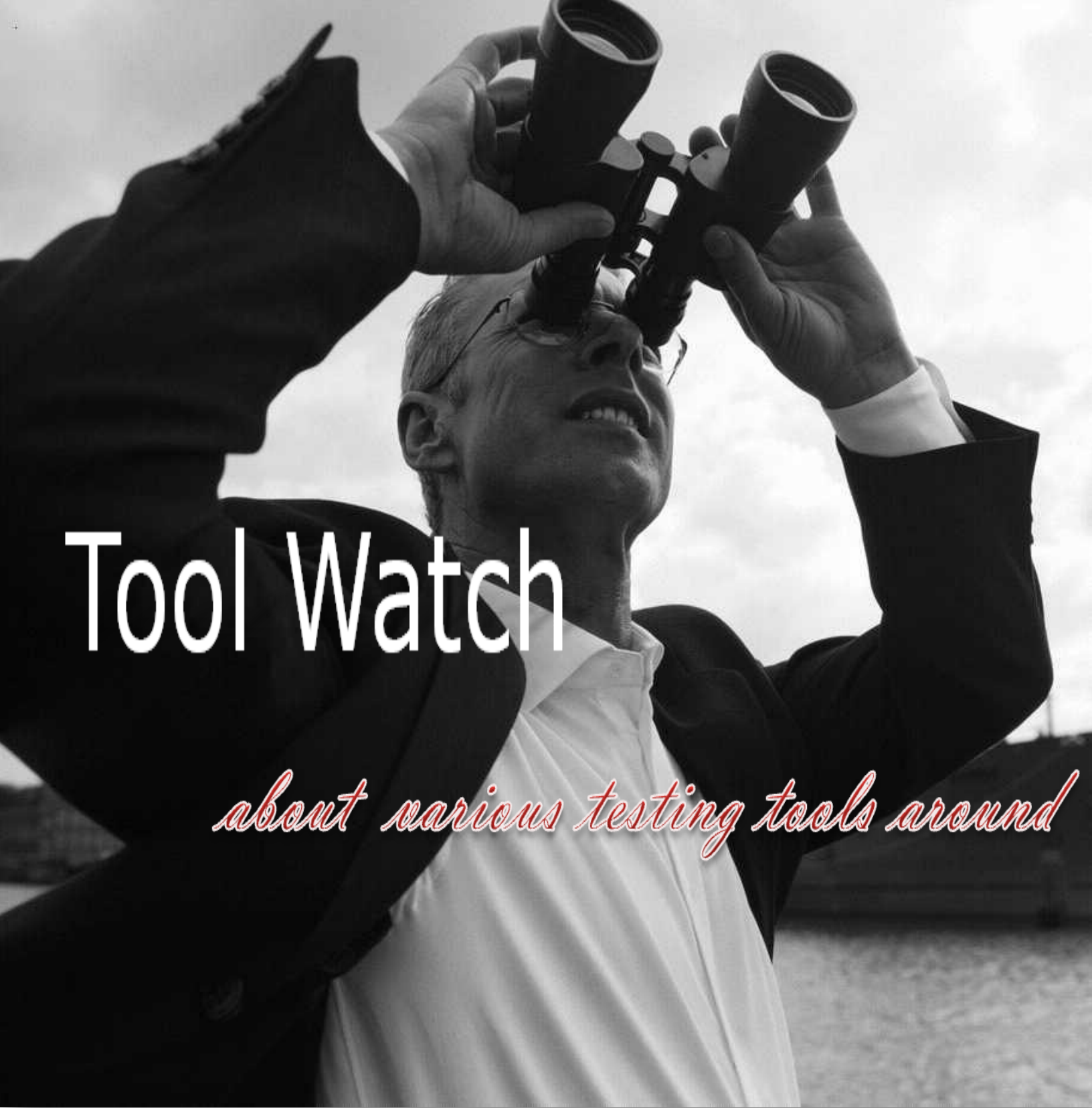
He can be reached at ash@stagsoftware.com.

Tickle your Testing Bone... 😊



© Copyright . Lalit Bhamare





Tool Watch

about various testing tools around

SAHI

By Chris Philip

One of the major challenges an automation tester faces is the **cross-browser testing**. With the increased demand from customers, many applications are getting developed in the format suitable to be compatible in almost all the popular browsers like Internet Explorer (now the trend is in IE8), Mozilla, Safari, Netscape, Chrome etc. Even though many testing tools boast about cross-browser support, many of them are having limitations in compatibility or support features.

Another challenge is in **reporting** of the test execution. Like all testing tools, QTP, the tool which I'm extensively using, provides a default report. HTML reporting is a solution for meeting up the requirement of extra details that the user demands to cover in test execution. This reporting can be customized. The HTML report that I designed used JQuery, Js and some complicated logics of data transfer. This is, however, not easy to modify when there is a change to be added.

A solution for both the above queries made me interested in **SAHI**.

SAHI is one tool which survives at least the level above the basic demands in cross-browser testing and that having a very clear and detailed reporting format. Easy debugging of codes is an added advantage.

Here you can find a brief on SAHI

Product Type: Web application functional and regression testing.

Target User: Functional and regression testers



IT Problem: Increasingly enterprises are deploying web applications to drive competitive advantage. As expectations for availability, speed, and reliability increase daily, ensuring the performance of web applications pre-production has become critical.

IT Download Description: SAHI is an automation and testing tool for web applications, with the facility to record and playback scripts. SAHI tool is developed in java and java script and uses java script to execute the browser events. This is a tool to test web applications. Some salient features include excellent recorder, platform and browser independence, no XPath, no waits, multithreaded playback, excellent Java interaction and inbuilt reporting. SAHI tool injects java script into web pages using a proxy and the java script helps in automating the web applications.

SAHI is a tester friendly tool which clears out most difficulties that testers face while automating web applications.

Intuitive interface and advanced wizards support:

- **Browser and Operating System independent** – SAHI achieves this feature by having separate UI for each browser. Also the code recorded from one browser works in any other browser.
- The above mentioned point reveals that SAHI got **powerful recorder which works across browsers.**
- **Powerful Object Spy** – Object spy is used to detect the object properties in the browser window. SAHI's object spy is powerful enough to capture almost all types of commonly present objects.
- **Intuitive and simple APIs**- the API got simple interfaces that even a beginner can interact fluently with tool.
- **Java script** based scripts for good programming control
- Version Controllable text-based scripts
- **In-built reports** and **multithreaded or parallel playback of tests** – The reports generated by the tool is clear and detailed. Time consumed in execution of each command will be provided. This enables to test an aspect of performance testing – time consumption in page loading, submission etc.
- Tests do not need the browser window to be in focus
- **Command line and ant support** for integration into build processes -
- **Supports external proxy**, HTTPS, 401 & NTLM authentications
- **Supports browser pop-ups and modal dialogs**
- **Supports AJAX** and highly dynamic web applications – This is one distinct advantage SAHI is having over existing leading testing tools. Lack of support to Ajax is one of the major problems faced by many testing tools.
- **Scripts are very robust** - No wait statements required even for AJAX and page loads
- Works on applications with random auto-generated ids
- Very lightweight and scalable
- Supports data-driven testing. Can connect to database, Excel or CSV file.
- Ability to invoke any Java library from scripts.

Terms of License: This is an open source tool which won the "**BEST OPEN SOURCE FUNCTIONAL AUTOMATED TEST TOOL among the Annual ATI (Automated Testing Institute) Honors.**"



Learn SAHI in 7 steps:

1. Pre requisites

Java 1.5 or above is needed for running SAHI.

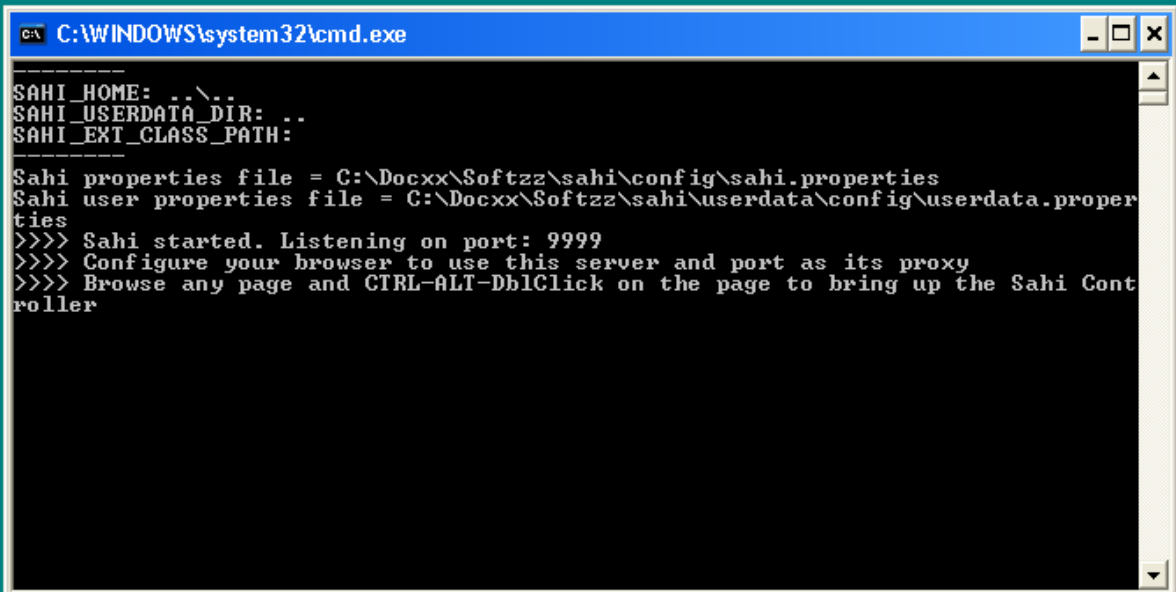
1. Download SAHI

Download SAHI V3 (2010-11-03) from SourceForge

2. Install SAHI

Unzip sahi.zip to a desired location

3. Start SAHI proxy:



```
C:\WINDOWS\system32\cmd.exe

SAHI_HOME: ..\..
SAHI_USERDATA_DIR: ..
SAHI_EXT_CLASS_PATH:

Sahi properties file = C:\Docxx\Softzz\sahi\config\sahi.properties
Sahi user properties file = C:\Docxx\Softzz\sahi\userdata\config\userdata.properties
>>>> Sahi started. Listening on port: 9999
>>>> Configure your browser to use this server and port as its proxy
>>>> Browse any page and CTRL-ALT-DblClick on the page to bring up the Sahi Controller
```

Windows: – Go to <sahi_root>\userdata\bin and run start_sahi.bat

Linux – Go to <sahi_root>/userdata/bin and run start_sahi.sh

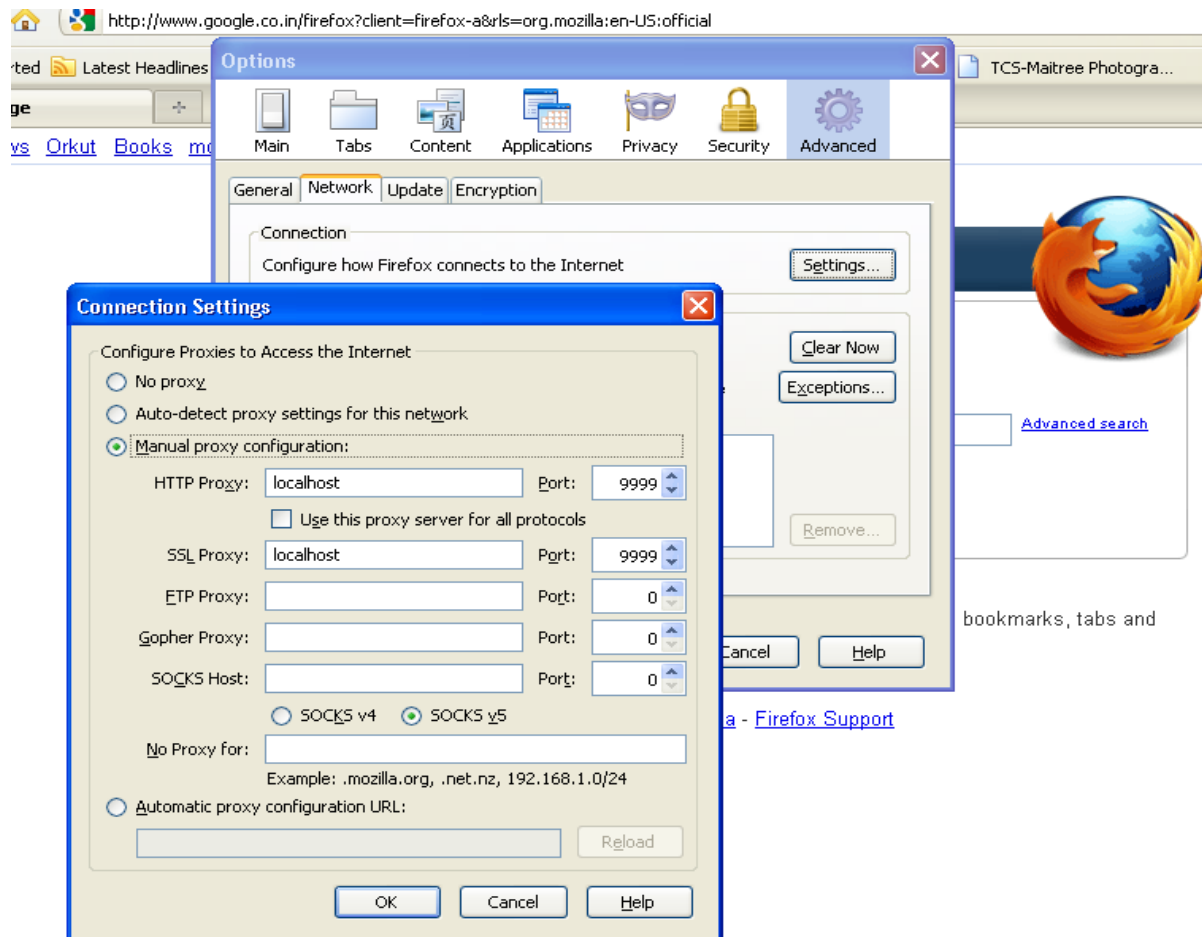
Note that by default SAHI uses port 9999. This can be modified through sahi.properties

4. Configure the browser:

You need to change your browser's proxy setting to use SAHI proxy.



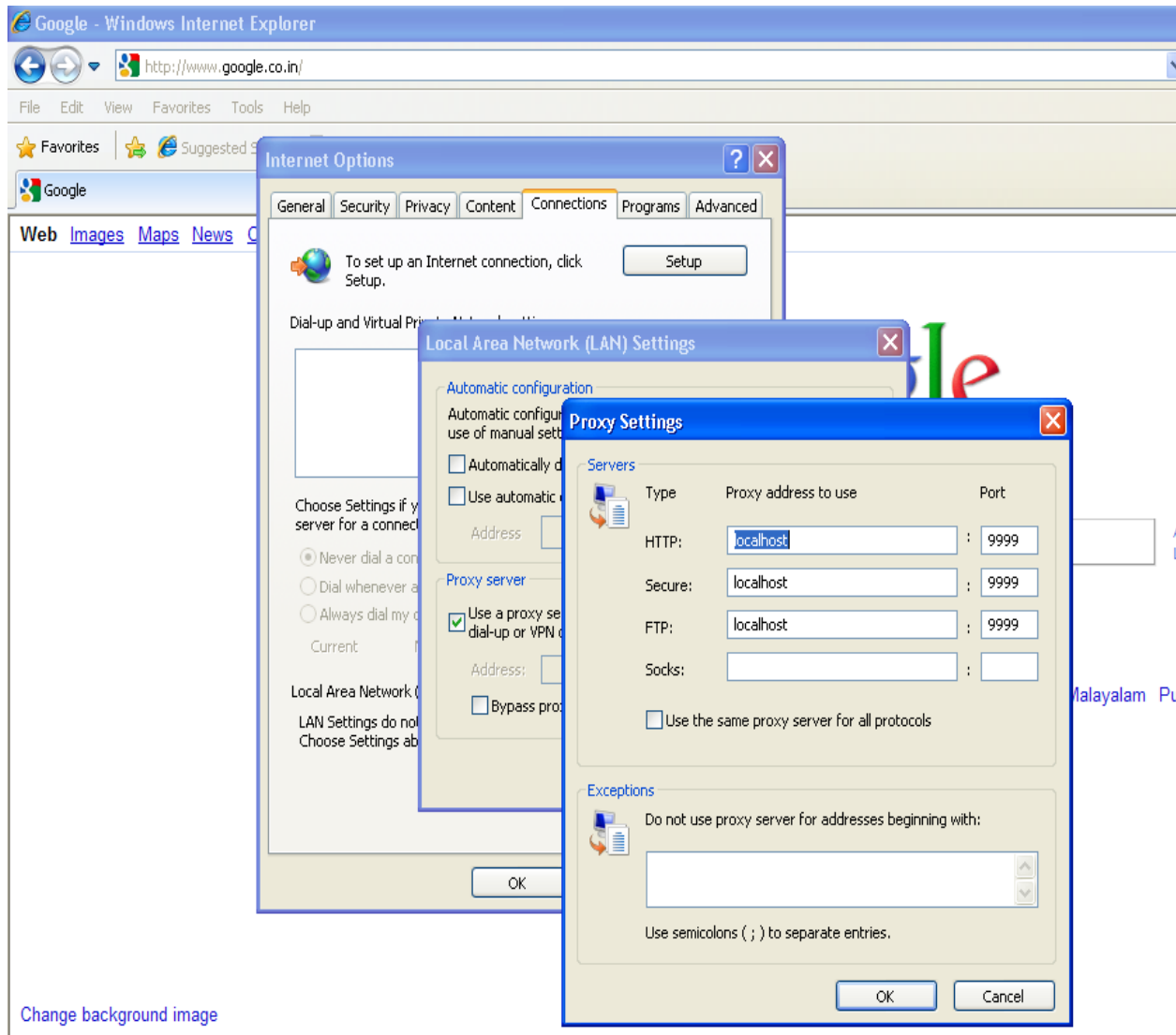
4.1 Firefox:



- Go to Tools > Options > General > Connection Settings >
- Set to "Manual Proxy Configuration"
- Set "HTTP Proxy" to "localhost"
- Set "Port" to "9999".
- Do the same for "SSL Proxy" too, if you wish to record and playback https sites
- Keep "Use the same proxy for all protocol" unchecked as SAHI does not understand protocols other than HTTP
- NOTE: "No Proxy for" should NOT have localhost in it.



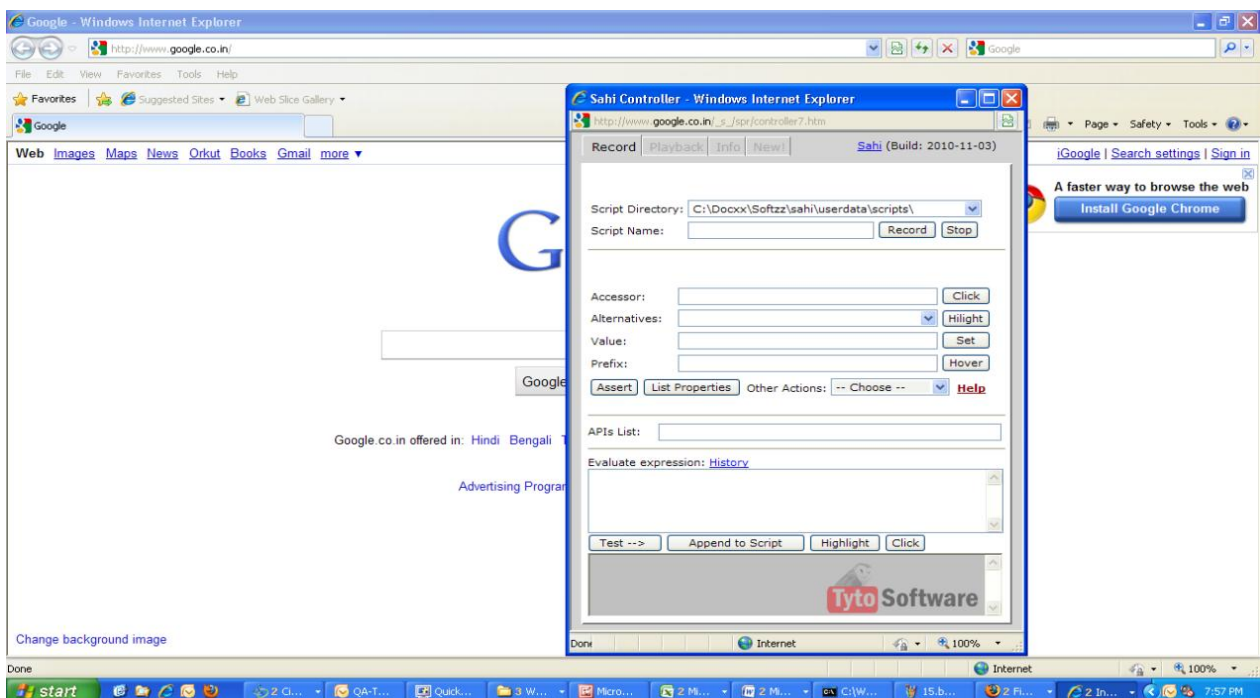
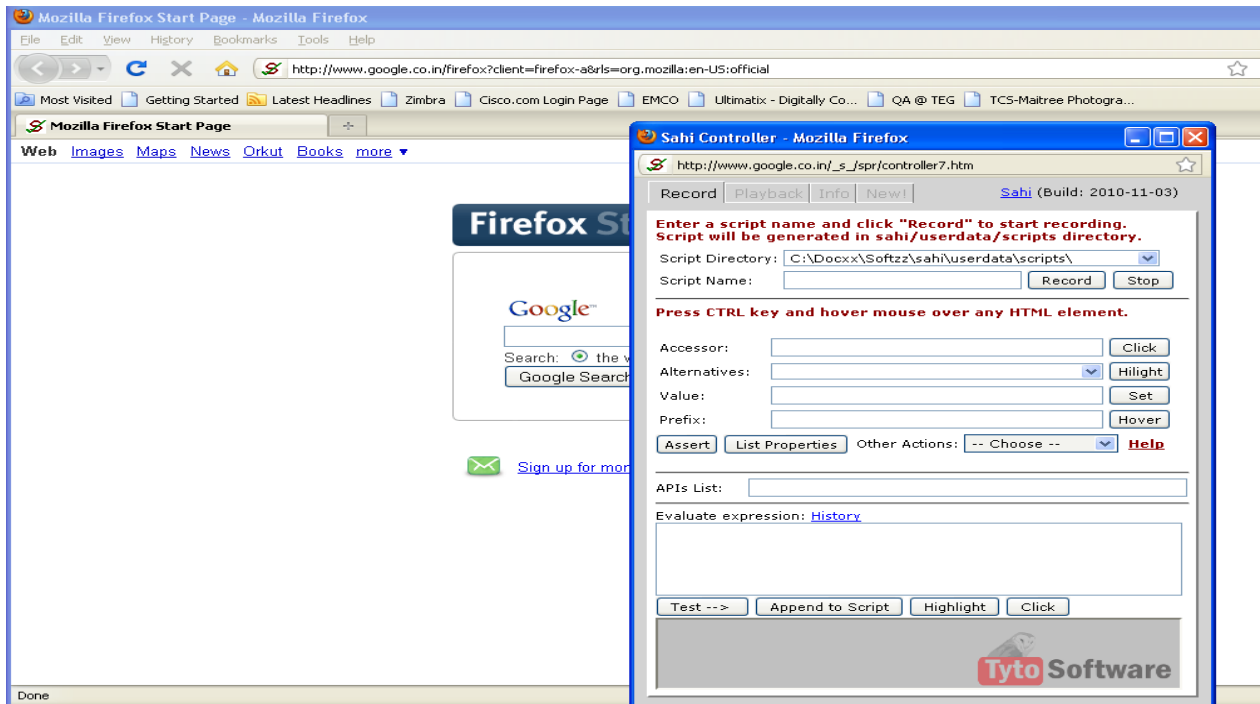
4.2 Internet Explorer:



- Go to Tools > Internet Options > Connections > LAN Settings >
- In "Proxy server" section, Check "Use a proxy server for your LAN"
- Click on "Advanced"
- For HTTP: set "Proxy address to use" to "localhost" and set "Port" to "9999"
- For Secure: set "Proxy address to use" to "localhost" and set "Port" to "9999" (if you want to test https sites too)
- Clear out anything in "Do not use proxy server for addresses beginning with:"
- Leave "Bypass proxy server for local addresses" unchecked
- Click OK



5. Recording through SAHI



- Press ALT and double click on the window which you want to record. Sahi's Controller window will pop up. If that does not work, press CTRL and ALT keys together and then double click. Make sure popup blockers are turned off.
- On the Controller, go to the Record tab (would be selected by default).



- Give a name for the script, and click 'Record'
- Navigate on your website like you normally would. Most actions on the page will now get recorded.
- Add an assertion:
 - Move the mouse over any html element while pressing Ctrl key. The Accessor field will get populated in the Controller.
 - Click the "Assert" button to generate assertions for the element. They will appear in the "Evaluate Expression" box.
 - Click "Test —>" to check that the assertions are true. You can evaluate any javascript using "Evaluate Expression" and "Test —>". Actions performed via the Controller will not be automatically recorded. Only actions performed directly on the page are automatically recorded. This lets you experiment on the webpage at recording time without impacting the script.
 - Once satisfied, click on "Append to Script". This will add the assertions to the Script.
- Click "Stop" to finish recording.

Note that the Controller can be closed and reopened at any time, without disrupting recording.

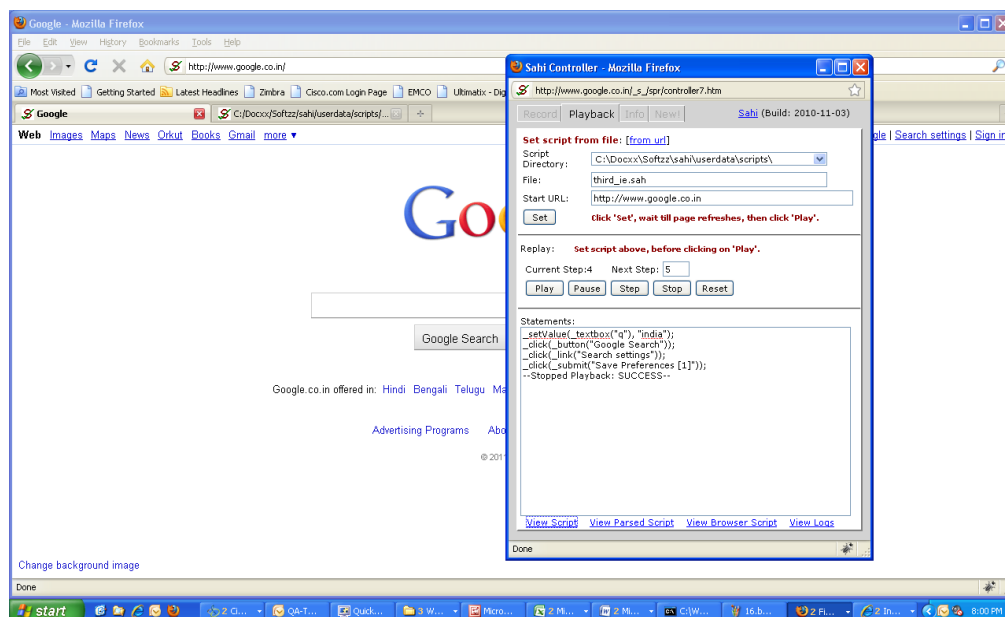
The API is ver simple that it can be handled by a new user without any confusion.

6.1 View the recorded script

The recoded script can be viewed and edited by opening the .sah file in the <sahi_root>\userdata\scripts directory. SAHI scripts are simple text files which use Javascript syntax.

The script can be edited even while recording, so that logical segregation into functions etc. can be done as recording happens.

6.2 Playing back the recorded script



- Open the SAHI Controller (CTRL-ALT-DbIClick on the page).

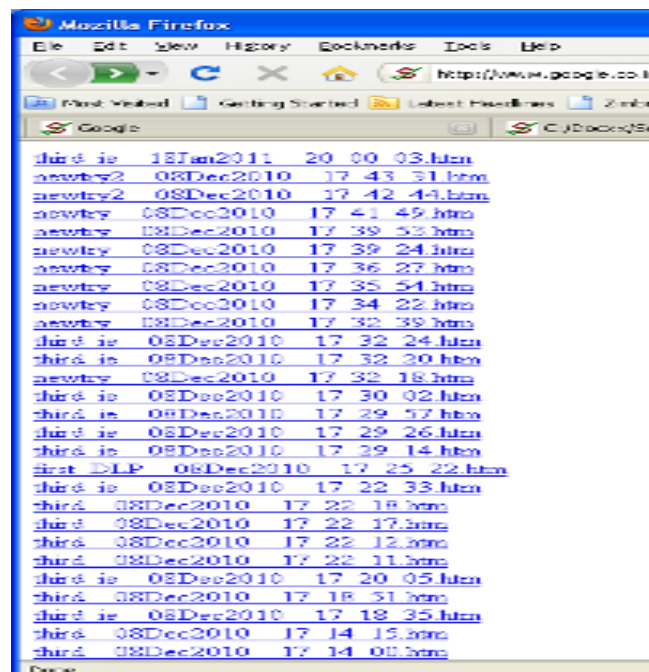


- Enter the script name in the "File:" field (with the help of the autocompletion feature).
- Enter the Start URL of the test. If you had started recording from <http://www.google.co.in>, use that URL.
- Click 'Set'.
- Wait for the page to reload.
- Click 'Play'.

Steps will start executing, and the Controller will be updated accordingly. Once finished, SUCCESS or FAILURE will be displayed at the end of the steps.

Note that the Controller can be closed at any time, without disrupting playback.

6. Viewing Logs



On the Controller, go to Playback tab and click on "View Logs" link at the bottom right. It will open a window with the results neatly formatted in HTML.

Clicking on a line in the logs will drill down to exact line in script.

You can also view the logs at <http://localhost:9999/logs>

Logs are one of the best features of Sahi. Being a dedicated QTP user, I go for an extra format of HTML reporting along with the default QTP report to get a clear picture of test execution. The HTML report requires additional coding in VBScript, which is not easily manageable by a fresher in the tool. In that aspect, SAHI is a bit forward in providing a detailed report of execution. Also the re-direction for log pages to the report page and further to the script, is an amazing and time saving feature.

Clicking on the desired log details will open the report corresponding to the log of execution selected.

This report consists of stepwise status(Success/ Failure), Number of Steps, Success Rate and most importantly, the Time Taken (and that too step-wise data is available).

Clicking on a step in report will re-direct the user to the corresponding step in the script. This helps in easy decoding, error-identification and thus in turn saves considerable amount of time.

Test	Total Steps	Failures	Errors	Success Rate	Time Taken (ms)
third_ie sah	4	0	0	100%	6500

Starting script

```
_setValue(_textbox("q"), "india"); at Jan 18, 2011 17:29:14  
click(_button("Google Search")); at Jan 18, 2011 17:29:14  
click(_link("Search settings")); at Jan 18, 2011 17:29:14  
click(_submit("Save Preferences [1]")); at Jan 18, 2011 17:29:14
```

Stopping script

```
_setValue(_textbox("q"), "india");  
click(_button("Google Search"));  
click(_link("Search settings"));  
click(_submit("Save Preferences [1]"));
```

So....That is it! You have successfully recorded and played back a SAHI script!

Running a test from command line

There are sample scripts available in sahi/userdata/bin directory for running tests on various browsers.

Command:

ff.bat <sahi_file|suite_file> <start_url>

Eg.

cd sahi/userdata/bin

ff.bat demo/demo.suite http://sahi.co.in/demo/

ff.bat demo/sahi_demo.sah http://sahi.co.in/demo/

Similar executables for other browsers exist as ie.bat, chrome.bat etc.
(You may need to adjust the browser paths in the batch files/shell scripts)



Biography



Chris Philip has been working with his first employer TCS since 10 months, passionately in area of Automation Testing. The passion and interest in automation was revealed during his college projects in robotics and successful completion of project work from India Space Research Organization, automating the microscopic and sensitive calculations regarding the minute changes in accelerometer data in launch vehicles, rockets and spacecrafts. The project was done in microprocessor programming language. He is an active member of Linux club, IEEE and Computer Society of India (CSI).

Special interest is software automation, having extensive hands-on experience in QTP, Sahi, Selenium and RFT.

Actively participates in blogs and discussions related to automation practices. Has presented 3 white papers till date about the automation practices, short cuts and interesting logics applicable in Quick Test Professional.

Chris is reachable at

christhomsonphilip@gmail.com

& on twitter [@chris_cruizer](https://twitter.com/chris_cruizer)

Chris has shared what he knows about...

Do you think even you've got something to share..?

Feel free to share your knowledge.

write to us at teatimewithtesters@gmail.com



next ISSUE

THE SCRUM PRIMER - PART 2

Managing multiple passions (by Anuj Magazine)

“Beautiful Code” read by a Testers (by Subhodip Biswas)

T' talks

Tool Watch and much more....



Our Team

❖ Founders and Editors:

Lalitkumar Bhamare & Pratikkumar Patel

❖ Editorial:

Lalitkumar Bhamare

❖ Magazine design and layout:

Lalitkumar Bhamare | cover-page picture: Maryse Coziol

❖ Mascot (BUG-BOSS) design:

Juhi Verma

❖ T' talks:

T Ashok

❖ Tech-Team :

Subhodip Biswas and Chris Philip

For feedback and to subscribe the ezine (FREE)
mail us at teatimewithtesters@gmail.com

Join our community on

facebook

follow us on



www.teatimewithtesters.com