THE INTERNATIONAL MONTHLY ON SOFTWARE TESTING

Tea-time with Testers

Jerry Weinberg
Observing and Reasoning about Errors

T Ashok
How Many Hairs do You have on Your Head?

Curtis Stuehrenberg
Why do We Need to Manage Our Bugs?

Anurag Khode
Using Nokia Energy Profiler

Neha Thakur
A Story of an Independent Testing Team

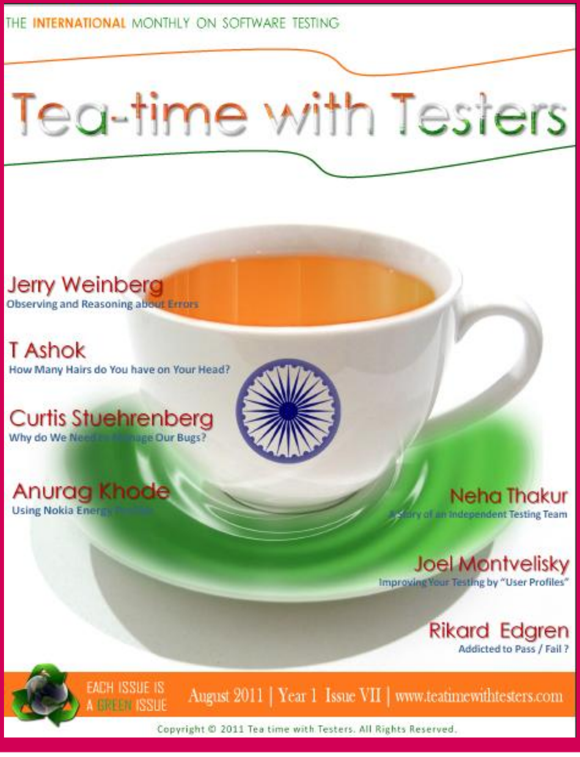
Joel Montvelisky
Improving Your Testing using "User Profiles"

Rikard Edgren
Addicted to Pass / Fail ?

EACH ISSUE IS A GREEN ISSUE

August 2011 | Year 1 Issue VII | www.teatimewithtesters.com

Editorial


THE INTERNATIONAL MONTHLY ON SOFTWARE TESTING
Tea-time with Testers
Jerry Weinberg
Observing and Reasoning about Errors
T Ashok
How Many Hairs do You have on Your Head?
Curtis Stuehrenberg
Why do We Need to Manage Our Bugs?
Anurag Khode
Using Nokia Energy Profiler
Neha Thakur
A Story of an Independent Testing Team
Joel Montvelisky
Improving Your Testing by "User Profiles"
Rikard Edgren
Addicted to Pass / Fail ?
EACH ISSUE IS A GREEN ISSUE
August 2011 | Year 1 Issue VII | www.teatimewithtesters.com
Copyright © 2011 Tea time with Testers. All Rights Reserved.

Created and Published by:

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

Editorial and Advertising Enquiries:

Email: teatimewithtesters@gmail.com
Pratik: (+91) 9819013139
Lalit: (+91) 9960556841

Dear Readers,

Apart from Software Testing, Social Science is another subject that attracts me most.

Looking at the bizarre events that are happening from last few months in India and also in other parts of the world, it's quite evident that what matters at the end of the day is proper governance and better management. In my opinion, there should be a strong will of a person to change what is wrong, unethical and what is dangerous in long run.

You won't be surprised if I connect this with our daily life as a Software Testing professional. When we talk about success or failure of any delivery, it does not solely depend on what programmer codes or what we testers test. Its major part lies in its good/bad planning, management and execution. In short, in its overall governance, doesn't it?

Why I am saying all this is because; it's just not enough to be a skilled tester. If you have the right attitude along with adequate testing skills, then there is nothing that can stop you from delivering your best. Good tester is not the one who just tests better. Good tester is one who also asks for what is right, helps others to understand what is right and struggles hard to eliminate the wrong.

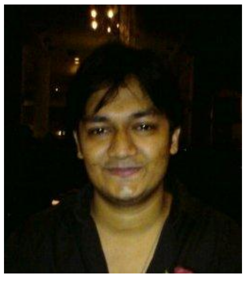
Good news is, we do have such people who are good testers in all aspects and have also contributed in this issue. Along with being good testers, they are good Directors, Managers and Leads as well. In this issue of Tea-time with Testers, we have tried our best to cover ***this*** aspect of a tester and I am sure you'll like all articles.

Special thanks to Curtis, T Ashok, Neha and Rikard for their contribution and highlighting those areas which we always wanted to talk about. Big thanks to Jerry Weinberg for mentoring our readers with his rich knowledge and experience.

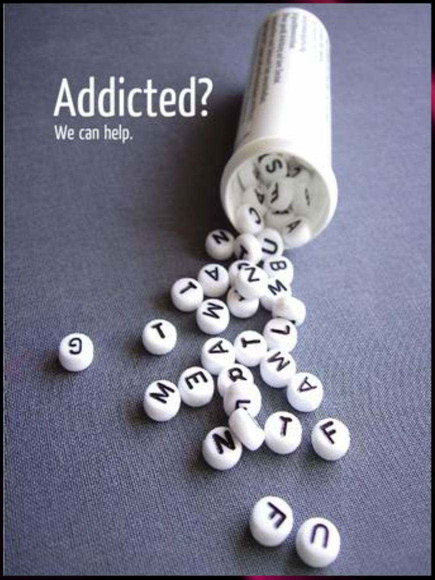
Enjoy Reading! Have a nice Tea-time!

Yours Sincerely,

[illegible] **Lalitkumar Bhamare**



QuickLook



Testing Puzzles
by Sebi

Crossword
by

QUALITY TESTING

CAST 2011 has shipped

One of the well-known Software Testing Conferences, **CAST 2011** concluded successfully on 10th of August this year. Publishing its experience report, written by **Jon Bach** who was also the chairperson of this conference:

By Jon Bach
on August 12, 2011

Association for Software Testing

"A few minor bugs, but it looks like the value far outweighed the problems.

James and I wanted it to be THE context-driven event, and it was — speakers from around the world, a tester-themed movie, tester games, a competition, lightning talks, emerging topics, half-day and full-day tutorials, an EdSig with Cem Kaner present on Skype, a live (3D) Weekend Testers Americas session, professional webcasts to the world, debates, panels, open season, facilitation, music, a real-time Skype coaching demo, real-time tweets, LIVE blogs posted as it happened (via http://typewith.me) and red cards for those who needed to speak NOW."

Read entire report on Jon's blog: Click **Here**!

CAST 2011
"Context-Driven Testing"
August 8 to 10, 2011
Lynnwood Convention Center - Seattle, WA, USA

uTest Doubles Down On Security Features, Partners With Veracode

by Mike Butcher
on August 14, 2011

In the wake of news recently that software and computers are being **compromised by hackers** and rogue states left right and centre (not to mention the recent attacks on Citibank and Sony) it's clearly going to make sense that your systems are well checked out, whether you're a large corporate or a startup.

So it's timely that software testing marketplace **uTest** is today expanding to offer security and localization testing services. That means it will have an end-to-end suite of testing services for web, desktop or mobile apps, adding to its existing services like functional, usability and load testing. uTest CEO, Doron Reuveni says the startup is aiming to become a 'one stop shop' for real-world testing. **READ MORE …**

Quality is delighting customers
QUALITY TESTING
Software Testing NEWS

For more updates on Software Testing, visit → Quality Testing - Latest Software Testing News!

Back To Index

Announcing…

Smart Tester of the Month Awards !!!

❖ Winner for Testing Crossword :
Ms. Sonal Agarwal , logicNEXT
From Noida - India

Sonal Agarwal

❖ Winners for Testing Puzzle :

Mr. Nishant Kumar Thakur
Adobe Systems India

Nishant Kumar Thakur

&

Mr. Srinivas Pattela
ValueLabs India

Srinivas Pattela

Congratulations !

Look Who's Rising

Seven Months

5000+ Readers

71 Countries

ONE Magazine

TEA-TIME WITH TESTERS

Subscribe here Right Away to get our all Issues for FREE

Observing and Reasoning About Errors (Part 1)

Men are not moved by things, but by the views which they take of them.- Epictetus

One of my editors complained that the first sections of this article series spend "an inordinate amount of time on semantics, relative to the thorny issues of software failures and their detection."

What I wanted to say is that "semantics" are one of the roots of "the thorny issues of software failures and their detection."

Therefore, I need to start this part of the series by clearing up some of the most subversive ideas and definitions about failure.

If you already have a perfect understanding of software failure, then skim quickly, and please forgive me.

1.1 Conceptual Errors About Errors

1.1.1. Errors are not a moral issue

"What do you do with a person who is 900 pounds overweight that approaches the problem without even wondering how a person gets to be 900 pounds overweight?" - Tom DeMarco

This is the question Tom DeMarco asked when he read an early version of the upcoming chapters. He was exasperated about clients who were having trouble managing more than 10,000 error reports per product. So was I.

Over thirty years ago, in my first book on computer programming, Herb Leeds and I emphasized what we then considered the first principle of programming:

The best way to deal with errors is not to make them in the first place.

In those days, like many hotshot programmers, I meant "best" in a moral sense:

(1) Those of us who don't make errors are better than those of you who do.

I still consider this the first principle of programming, but somehow I no longer apply any moral sense to the principle, but only an economic sense:

(2) Most errors cost more to handle than they cost to prevent.

This, I believe, is part of what Crosby means when he says "quality is free." But even if it were a moral question, in sense (1), I don't think that Pattern 3 cultures—which do a great deal to prevent errors—can claim any moral superiority over Pattern 1 and Pattern 2 cultures—which do not. You cannot say that someone is morally inferior because they don't do something they cannot do, and Pattern 1 and Pattern 2 software cultures, where most programmers reside, are culturally incapable of preventing large numbers of errors. Why? Let me put Tom's question another way:

"What do you do with a person who is rich, admired by thousands, overloaded with exciting work, 900 pounds overweight, and has 'no problem' except for occasional work lost because of back problems?"

Tom's question presumes that the thousand pound person perceives a weight problem, but what if they perceive a back problem. My Pattern 1 and 2 clients with tens of thousands of errors in their software do not perceive they have a serious problem with errors. They are making money, and they are winning the praise of their customers. On two products out of three, the complaints are generally at a tolerable level. With their rate of profit, who cares if a third of their projects have to be written off as a total loss?

If I attempt to discuss these mountains of errors with Pattern 1 and 2 clients, they reply, "In programming, errors are inevitable, but we've got them more or less under control. Don't worry about errors. We want you to help us get things out on schedule." They see no more connection between enormous error rates and two-year schedule slippages than the obese person sees between 900 pounds of body fat and pains in the back. Can I accuse them of having the wrong moral attitude about errors? I might just as well accuse a blind person of having the wrong moral attitude about the rainbow.

But it is a moral question for me, their consultant. If my thousand-pound client is happy, it's not my business to tell him how to lose weight. If he comes to me with back problems, I can show him through a diagram of effects how weight affects his back. Then it's up to him to decide how much pain is worth how many chocolate cakes.

1.1.2. Quality is not the same thing as absence of errors

Errors in software used to be a moral issue for me, and still are for many writers. Perhaps that's why these writers have asserted that "quality is the absence of errors." It must be a moral issue for them, because otherwise it would be a grave error in reasoning. Here's how their reasoning might have gone wrong. Perhaps they observed that when their work is interrupted by numerous software errors, they can't appreciate any other good software qualities. From this observation, they can conclude that many errors will make software worthless—i.e., zero quality.

But here's the fallacy: Though copious errors guarantee worthlessness, but zero errors guarantees nothing at all about the value of software.

Let's take one example. Would you offer me $100 for a zero defect program to compute the horoscope of Philip Amberly Warblemaxon, who died in 1927 after a 37-year career as a filing clerk in a hat factory in Akron? I doubt it, because to have value, software must be more than perfect. It must be useful to someone.

Still, I would never deny the importance of errors. First of all, if I did, Pattern 1 and Pattern 2 organizations would stop reading this book. To them, chasing errors is as natural as chasing sheep is to a German Shepherd Dog. And, as we've seen, when they see the rather different life of a Pattern 3 organization, they simply don't believe it.

Second of all, I do know that when errors run away from us, we have lost quality. Perhaps our customers will tolerate 10,000 errors, but, as Tom DeMarco asked me, will they tolerate 10,000,000,000,000,000,000,000,000,000? In this sense, errors are a matter of quality. Therefore, we must train people to make fewer errors, while at the same time managing the errors they do make, to keep them from running away.

1.1.3. The terminology of error

I've sometimes found it hard to talk about the dynamics of error in software because there are many different ways of talking about errors themselves. One of the best ways for a consultant to assess the software engineering maturity of an organization is by the language they use, particularly the language they use to discuss error. To take an obvious example, those who call everything "bugs" are a long way from taking responsibility for controlling their own process. Until they start using precise and accurate language, there's little sense in teaching such people about basic dynamics.

Faults and failures. First of all, it pays to distinguish between failures (the symptoms) and faults (the diseases). Musa gives these definitions:

A failure "is the departure of the external results of program operation from requirements."

A fault "is the defect in the program that, when executed under particular conditions, causes a failure."

For example: An accounting program had an incorrect instruction (fault) in the formatting routine that inserts commas in large numbers such as "$4,500,000". Any time a user prints a number greater than six digits, a comma may be missing (a failure). Many failures resulted from this one fault.

How many failures result from a single fault? That depends on

- the location of the fault
- how long the fault remains before it is removed
- how many people are using the software.

The comma-insertion fault led to millions of failures because it was in a frequently used piece of code, in software that has thousands of users, and it remained unresolved for more than a year.

When studying error reports in various clients, I often find that they mix failures and faults in the same statistics, because they don't understand the distinction. If these two different measures are mixed into one, it will be difficult to understand their own experiences. For instance, because a single fault can lead to many failures, it would be impossible to compare failures between two organizations that aren't careful in making this "semantic" distinction.

Organization A has 100,000 customers who use their software product for an average of 3 hours a day. Organization B has a single customer who uses one software system once a month.
Organization A produces 1 fault per thousand lines of code, and receives over 100 complaints a day. Organization B produces 100 faults per thousand lines of code, but receives only one complaint a month. Organization A claims they are better software developers than Organization B.
Organization B claims they are better software developers than Organization A. Perhaps they're both right.

The System Trouble Incident (STI). Because of the important distinction between faults and failures, I encourage my clients to keep at least two different statistics. The first of these is a data base of "system trouble incidents," or STIs. In this book, I'll mean an STI to be an "incident report of one failure as experienced by a customer or simulated customer (such as a tester)." I know of no industry standard nomenclature for these reports—except that they invariably take the form of **TLAs (Three Letter Acronyms)**. The TLAs I have encountered include:

- STR, for "software trouble report"
- SIR, for "software incident report," or "system incident report"
- SPR, for "software problem report," or "software problem record"
- MDR, for "malfunction detection report"
- CPI, for "customer problem incident"
- SEC, for "significant error case,"
- SIR, for "software issue report"
- DBR, for "detailed bug report," or "detailed bug record"
- SFD, for "system failure description"
- STD, for "software trouble description," or "software trouble detail"

I generally try to follow my client's naming conventions, but try hard to find out exactly what is meant. I encourage them to use unique, descriptive names. It tells me a lot about a software organization when they use more than one TLA for the same item. Workers in that organization are confused, just as my readers would be confused if I kept switching among ten TLAs for STIs.

The reasons I prefer STI to some of the above are as follows:

1. It makes no prejudgment about the fault that led to the failure. For instance, it might have been a misreading of the manual, or a mistyping that wasn't noticed. Calling it a bug, an error, a failure, or a problem, tends to mislead.

2. Calling it a "trouble incident" implies that once upon a time, somebody, somewhere, was sufficiently troubled by something that they happened to bother making a report. Since our definition of quality is "value to some person," someone being troubled implies that it's worth something to look at the STI.

3. The words "software" and "code" also contain a presumption of guilt, which may unnecessarily restrict location and correction activities. We might correct an STI with a code fix, but we might also change a manual, upgrade a training program, change our ads or our sales pitch, furnish a help message, change the design, or let it stand unchanged. The word "system" says to me that any part of the overall system may contain the fault, and any part (or parts) may receive the corrective activity.

4. The word "customer" excludes troubled people who don't happen to be customers, such as programmers, analysts, salespeople, managers, hardware engineers, or testers. We should be so happy to receive reports of troublesome incidents before they get to customers that we wouldn't want to discourage anybody. Similar principles of semantic precision might guide your own design of TLAs, to remove one more source of error, or one more impediment to their removal. Pattern 3 organizations always use TLAs more precisely than do Pattern 1 and 2 organizations.

System Fault Analysis(SFA). The second statistic is a database of information on faults, which I call SFA, for System Fault Analysis. Few of my clients initially keep such a database separate from their STIs, so I haven't found such a diversity of TLAs. Ed Ely tells me, however, that he has seen the name RCA, for "Root Cause Analysis." Since RCA would never do, the name SFA is a helpful alternative because:

1. It clearly speaks about faults, not failures. This is an important distinction. No SFA is created until a fault has been identified. When a SFA is created, it is tied back to as many STIs as possible. The time lag between the earliest STI and the SFA that clears it up can be an important dynamic measure.

2. It clearly speaks about the system, so the database can contain fault reports for faults found anywhere in the system.

3. The word "analysis" correctly implies that data is the result of careful thought, and is not to be completed unless and until someone is quite sure of their reasoning. "Fault" does not imply blame. One deficiency with the semantics of the term "fault" is the possible implication of blame, as opposed to information. In an SFA, we must be careful to distinguish two places associated with a fault, either of these implies anything about whose "fault" it was:

a. origin: at what stage in our process the fault originated
b. correction: what part(s) of the system will be changed to remedy the fault Pattern 1 and 2 organizations tend to equate these two notions, but the motto, "you broke it, you fix it," often leads to an unproductive "blame game." "Correction" tells us where it was wisest, under the circumstances, to make the changes, regardless of what put the fault there in the first place. For example, we might decide to change the documentation—not because the documentation was bad, but because the design is so poor it needs more documenting and the code is so tangled we don't dare try to fix it there.

If Pattern 3 organizations are not heavily into blaming, why would they want to record "origin" of a fault? To these organizations, "Origin" merely suggests where action might be taken to prevent a similar fault in the future, not which employee is to be taken out and crucified. Analyzing origins, however, requires skill and experience to determine the earliest possible prevention moment in our process. For instance, an error in the code might have been prevented if the requirements document were more clearly written. In that case, we should say that the "origin" was in the requirements stage.

to be continued in Next Issue ...

Back To Index

Biography

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

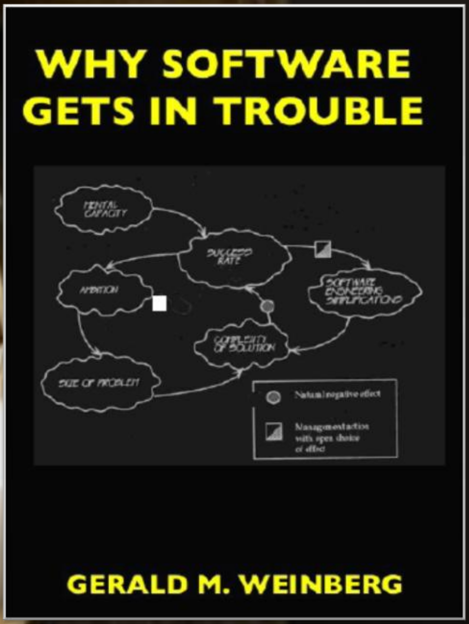
For more than 50 years, he has worked on transforming software organizations. He is author or co-author of many articles and books, including The Psychology of Computer Programming. His books cover all phases of the software life-cycle. They include Exploring Requirements, Rethinking Systems Analysis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

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

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

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

Why Software gets in Trouble is Jerry's world famous book.



WHY SOFTWARE GETS IN TROUBLE

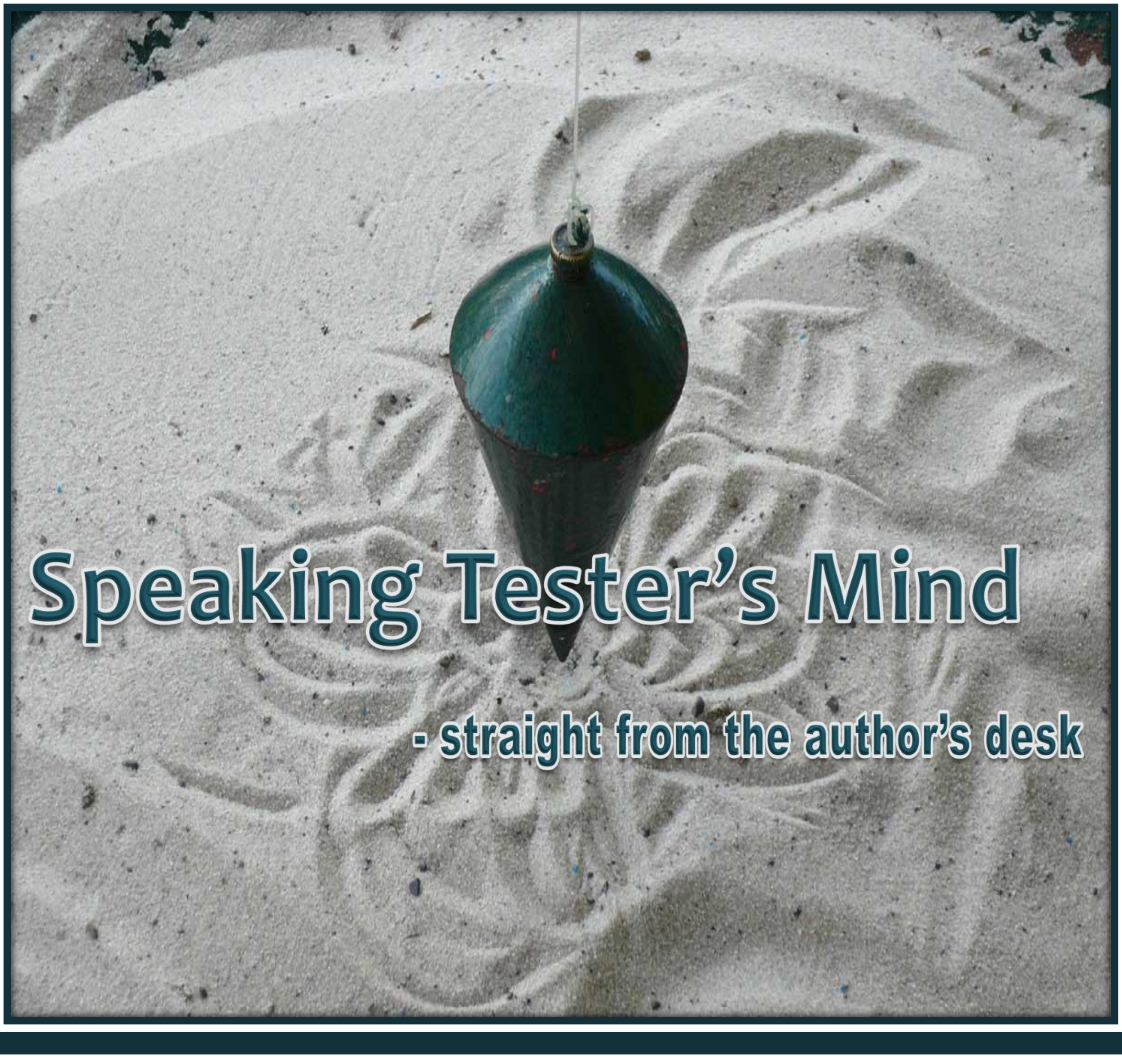
GERALD M. WEINBERG

TTWT Rating: ★★★★★

Many books have described How Software Is Built. Indeed, that's the first title in Jerry's **Quality Software Series**. But why do we need an entire book to explain Why Software Gets In Trouble? Why not just say people make mistakes? Why not? Because there are reasons people make mistakes, and make them repeatedly, and fail to discover and correct them. That's what this book is about.

Its sample can be read online here.

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


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

A story of...

...an Independent Testing Team !

By Neha Thakur

Before I start with this story, I would like to take help of two quotations below - one famous and one not so famous, which explain the importance of the independent testing team in today's era and at the same time, the responsibilities which come with this independence.

- *"Fundamentally you can't do your own testing; it's a question of seeing your own blind spots." ~Ron Avitzur*

- *"With great power comes great responsibility"~Spiderman*

Well, I am a crash-o-holic, in fact that is how I define myself and believe me I am really good at that ☺. I bet there will be very few developers (close to none) in the world who can crash or break their own created software/s. This in itself implies the importance of an independent test culture in any organization.

I have an interesting and a true story to share with you but with following disclaimer:

DISCLAIMER

All characters mentioned in this article are NOT at all fictitious
(though names changed).

Any resemblance to real persons, living or dead is nothing but mere fact.

Somewhere in TBD organization, Harry was working as a Quality lead, and was initially assigned to work with a project manager. All the testers were reporting the issues/defects/bugs directly to him. It was PM's choice to select a few bugs and make those few defects visible to the client. Harry discussed this issue with the manager and tried to find out his point of views, but from his past experiences he

already knew that what could probably be the reasons behind this, somewhere he was juggling with the thoughts, that, can it be time constraint, high project pressure from client surmounting or, simply, just to maintain a good face value for organization with minimum bugs and quality. There were numerous reasons explained in that discussion and to Harry's surprise one reason that was obvious to him later was that testers were subjugated because of a still existing ***ancient philosophy{myth}*** that developers are smarter than testers. To ease this volatile situation, there was another internal project created and all the defects which were rather not important from PM's point of view were moved to this project. For some time, attention was given to these issues but when client started logging issues, they became the priority, and now the issues logged by quality team at offshore took a backseat and were never attended.

The testing team was an enthusiastic one but when they came to know that there is no action taken upon their findings and at the same time client also was unaware of those issues, they were disheartened. Then came the inflammable discussion which spurred a sequence of events to be followed, the topic was **"The need of an independent testing team"** which won't directly fall under authorization of any PM. But at the same time an important question was asked whether the testing team is capable and mature enough to make its own decisions.

Most importantly the quality leads/managers responsibility is to educate the colleagues and co-workers and seniors the importance of testing and how critical it is for a quality project release. How important it is to keep the test team morale high. Testing is an ART in itself and testers should be treated with mutual respect and everyone should understand the underlying philosophy is that they don't have any self-motive in logging issues rather it's the need of the hour and project quality that matters to them. The more transparent the system is the less complication will be there. To run a smooth and successful project you need both dev and test team having a strong mutual understanding and respect for each other's work. And this can only be done by understanding each other's work and being a helping hand to each other at times, right from studying the requirements together and finding gaps, query resolutions testers can share their test cases/scenarios with the dev team and at the same time testers can have an understanding of the technical reasons and how important/critical is the bug and this can enhance the bug reporting and gives Dev a better understanding of issues. Well, this in itself is a long topic and can be consume the entire topic. Moving on to the remaining Story, this continues hereafter…

After a long battle, finally the management concluded and the deserving side got justice☺ ! Then they started the formation of a highly anticipated "TESTING RULES".

The following guideline is an important and necessary course of action in order to have an efficient ***Independent Testing*** team. Again the thumb rule is, "There is no thumb rule" as every organization has its own needs and demands. In order to cater the demands, there are certain guidelines, which needs to be followed but the same guidelines may not make any sense to a different organization altogether.

In my opinion, Independent Testing Team is important because:

1. An independent testing team is an unbiased one and it has more credibility in finding crucial bugs which can be missed by developers and which can cause a major impact on the releases. At the same time it's not an understatement to say that team might miss some defects due to its lack of knowledge of the domain or product but this can be compensated by educating the team.

2. The independent test team is well equipped with the latest infrastructure and any special needs of the organization can be met in a very short turnaround time. At the same time you can easily find economically viable testing teams/organizations having a good expertise and are proficient in ferreting bugs.

3. Developers can be more productive and will have a more focused approach in improving the product/software as their work load will be lessened by an equally productive team whose intent will also be towards delivering a high quality product.

4. The prerogative of the test team will strictly adhere to achieving client's business goals and objectives and will not be deflected in case of changing schedules or the project management activities.

5. The relationship between the development and the testing team plays a very crucial role in the effectiveness of the testing, if not handled and guided properly it can lead to a new **"STAR WARS"**. When the tester is filing bugs, they should understand the technical limitations and should have at least good if not strong technical skills in order to better understand the software and at the same time may help developers in finding possible solutions and should provide detailed defects summary and reproduction steps always. While at the same time, developers should not take the amount of defects logged against their modules personally and should motivate the testers as well.

6. The outputs are subjected to the proper analysis and minimal exposure of risks and may in turn be advantageous and end up giving higher ROI, better customer satisfaction as testing is performed from end user point of view, less time to market and more process oriented approach followed.

Discuss this topic on Quality Testing
Click HERE

Back To Index

Biography

Neha Thakur is passionate experienced software testing professional whose written articles and presentations have received awards in many internal conferences.

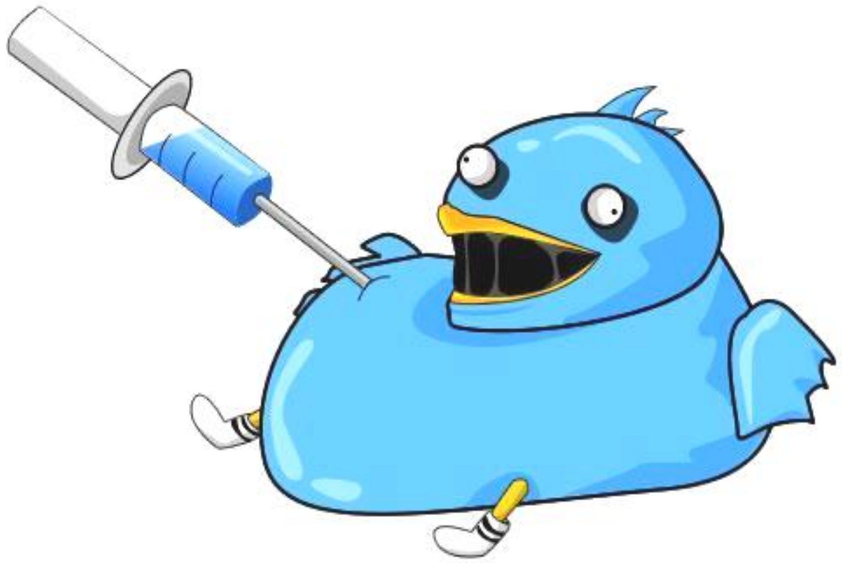
she is a regular writer for the QAI International Software testing conference since 2008 and she has also spoken at the CAST 2009 conference in Colorado.

Neha is an engineering honors graduate and is currently working as QA Lead at Diaspark INC, Indore, and has previously worked as Business Tech Analyst with Deloitte Consulting and as Software Engineer with Patni Computer Systems Pvt Ltd.

Her interests and expertise include test automation, agile testing and creative problem solving techniques.

Neha is an active and enthusiast speaker on any topic related to testing and has presented papers on various topics at STC in last few years.

Addicted to Pass / Fail ?



By Rikard Edgren

I just use it; there are no disturbing effects… (Anonymous addict)

Many years ago, people in software testing thought that complete testing was possible. They believed everything important could be specified in advance; that the necessary tests could be derived; and then check if the outcome was positive or negative.

Accelerated by the binary disease this thinking made its way into many parts of the testing theory, and is now adapted unconsciously by most testers.

We now know that testing is a sampling activity, and that communicating what is important is the fundamental skill, but what started as a virus, has become an addiction very difficult to get rid of.

Symptoms

- The most important thing to do after a test, is to set a Pass or a Fail in the test management system
- You feel a sense of accomplishment when you set Pass/Fail and move on to the next test
- You spend time and energy reporting Pass/Fail, terminology that don't mean a lot
- You structure your test in un-productive ways, just so Pass/Fail can be used
- You reduce the value of requirements documents by insisting everything must be verifiable
- You think testing beyond requirements is very difficult
- You count Passes and Fails, but don't communicate what is most important
- You don't consider Usability when testing, even though you'd get information almost for free
- You haven't heard of serendipity
- Status reporting is easy since counting Pass/Fail is the essence

Cause

Software testing is suffering a binary disease, with the Pass/Fail as the most rooted and wide-spread problem. Once in use, it provides quantification that people believe are objective and useful for rational decision.

We make software for computers, and use computers for planning, execution and reporting. Our theories reflect this much more, way much more than the fact that each piece of software is unique, made for humans, by humans.

If Gigerenzer's tools-to-theories heuristic is applicable - the theories we build are based, and accepted, depending on the tools we are using - this behavior will stay for a long time.

Cure

It is easy to say "Just stop!", but it doesn't help addicts. And intense hands-on adjustments by a trained doctor doesn't scale.

But I think this is something you can grow out of step by step (that's how I did it myself)

- Do some deviations when executing tests
- Look at some more places than what is stated in the Expected Results field
- Write the occasional test idea using the word "investigate"
- Put the numbers in smaller font in your status report
- Observe the software without a hypothesis to falsify
- See it as your daily medicine; eventually any Pass/Fail usage will seem ridiculous.

I told a friend about this and got the response: "*Well you know, for some types of requirements, it is very convenient to set Pass/Fail. I think I like the mix.*"

I felt sympathy; I know the feeling, have wandered in those dark woods for a long time; we'll fix this.

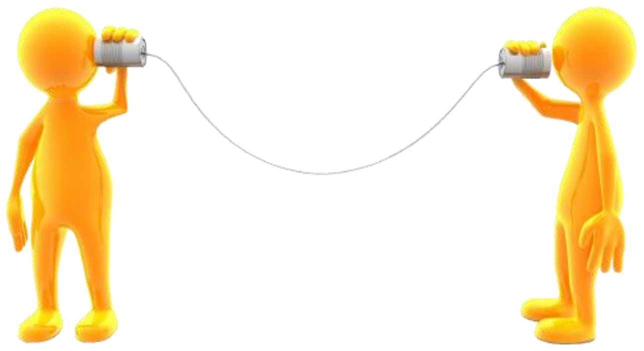
What's on the Other Side?

When you have liberated yourself, you can test much broader, you are not limited to singularities and binary conclusions. You can report the state of the product with understandable sentences, instead of numbers with a false sense of meaning.

You can communicate interesting things you found; risks and opportunities; problems and benefits; fears and rumors you have killed.

You can find new test ideas and approaches, you can ask richer questions than *is this correct or not*? You can learn things, and grow as tester.

Communicating What's Important



So what can you do in an environment where management demands numbers, and even could interpret/manipulate information so the project is seemingly healthy?

To start with you should find out what they really need, and why the testing is taking place. Talk to managers about information objectives; say that you want more guidelines than the requirements. Discuss which quality characteristics the testers should have in the back of their head all the time. Ask what they are afraid can happen when the product is launched.

You should be able to provide different information to different people, in a timely manner. Some only need to see the bug reports, some want a weekly meeting with a summary, many need to know they can trust you, and you accomplish that by finding out important things about the software.

It is easy to do an aggregation of Pass/Fail and make it look like a status report summary. But the art of communicating the vital information is something you need to practice; you need knowledge about details, and the ability to summarize and explain in a variety of manners.

As training, you can end each day with a 15 seconds recap about today's findings, and a 30 second summary of the health of the overall project.

Checking is binary, testing is not !

Back To Index

Biography

Rikard Edgren is a humanistic tester since 1998.

He is specialized in generalities like test analysis and exploratory testing.

Rikard is also a member of the think-tank **The Test Eye**.

A Voice on "Teach-Testing" Campaign!

Well, Software Testing as a separate course in universities is definitely a good idea to let the student know its importance in Software Industry.

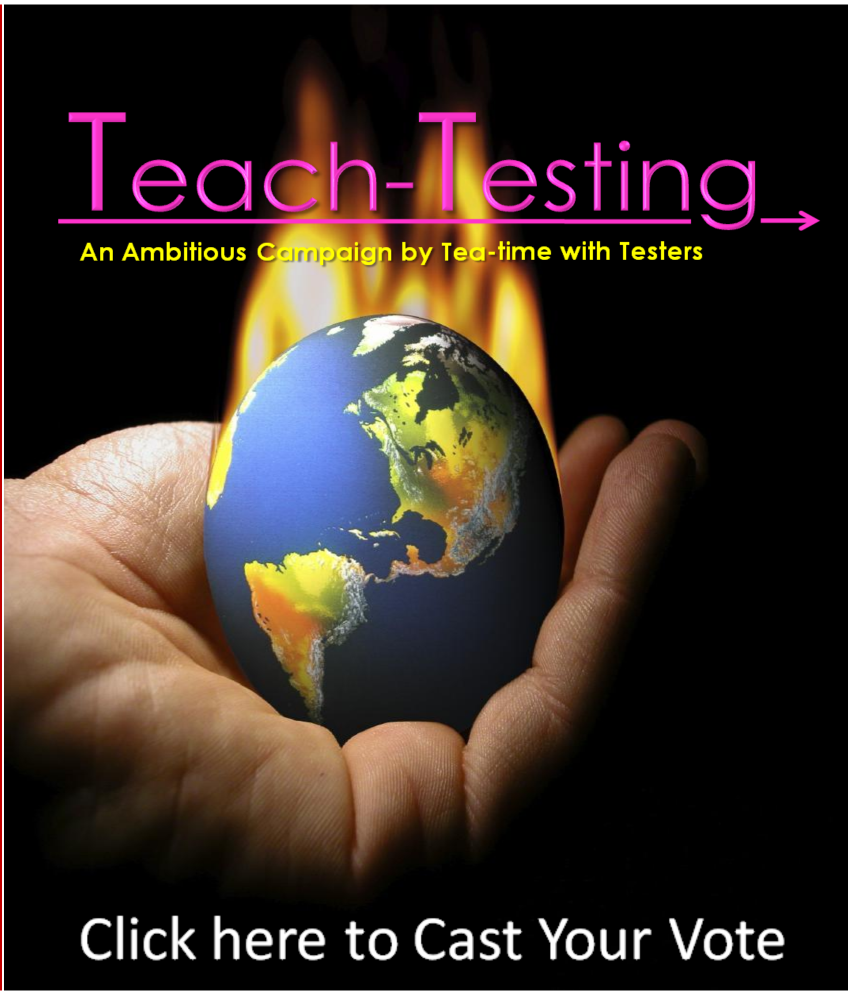
There was a time when I was pursuing my B.Tech degree. Software Testing was a single topic included in the Software Engineering subject.

I could not really recognize whether it's worth to be a part of Software Testing world. May be the lack of awareness or may be the bounded quantity of the subject matter. But it's time to open the box and make some space for the same.

I raise my vote in favour of the post.

Thumbs Up ! :-)

- Heena


Teach-Testing
An Ambitious Campaign by Tea-time with Testers
Click here to Cast Your Vote

Do you have any Questions or Feedback on articles that we publish in Tea-time with Testers?

No Problemo! We will publish your Feedback/Comments and also the answers to your Questions that you have for our Authors.

Do write us your Feedback and Questions in below format and send it to teatimewithtesters@gmail.com :

- Your Name
- Your Brief Introduction
- Article Name
- Your Feedback or Questions if any

Make sure to write **Feedback For < Article Name>** in your subject line.

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

Why do We Need to Manage Our Bugs ?

By Curtis Stuehrenberg

Testers are a strange group

We work in a very strange industry when we choose to make the leap into the dark chasm with no apparent bottom that is software testing. For most of the computer software industry, there is a clear line delineating "production" folk and "management" folk. Producers actually produce "things" be it software that can be sold, systems on which said software can live and thrive like binary sea monkeys, or hardware that provides a seemingly living shell encasing the magic voodoo bringing movies of pet birds singing "It's Getting Hot in Here" to amuse us. Managers sort through the different producers and their tasks making sure those "things" they're working on are produced as requested, as expected, within a certain time frame, and at the smallest cost possible. Do you see the problem? Testers don't fit neatly into either category. We don't necessarily manage or oversee the work of people actually producing things, yet we don't actually produce "things" in the same way we can say a database administrator produces a schema. We often swap activities between ensuring things are done as expected and producing things for other to consume. We're sort of hybrids[1] spanning the two worlds, a state which has left us with some identity problems. Are we producers or are we managers?

The debate seems to have pushed a few concepts forward that in my opinion need a little thought and review. One of these is defect management, bug tracking, issue control, or whatever it is your organization calls it. But what does bug management and do bugs in general have to do with our status as testers? They matter because bugs are a nice little piece of product. Sure they might be solely information, but so is software. Bugs have properties. Bugs have process. Bugs can be counted up and tallied and analyzed in all sorts of magical and ways. Bugs even have nice little consequential products that result from being tallied and analyzed, products usually called "metrics" or "reports." Bugs feel like "something" and so are thought of as a "product" making us a producer rather than a manager or nothing since we don't really manage anything.

Finding bugs gives many testers a sense of purpose and value on a product. The act of recording bugs somewhere, submitting them to analysis, assigning them out to someone, verifying them when corrected, and finally archiving them off somewhere for later review is therefore as close to a sacrosanct idea as we probably get in a profession attracting an unusual number of iconoclasts. It's not something usually questioned, and heaven knows I didn't until a few years ago.

This is a True Story … Honest

What changed for me was working on an agile project being steamrolled/lead by a very experienced giant killer sort of fellow named Max Guernsey.[2] At the time I was ostensibly employed as the quality assurance manager for the entire company. Of course seeing that I was the "QA Manager," everyone complained to me about what they perceived as a lack of quality on Max's project team. No one complained to Max because he can be a very intimidating and intelligent force of nature, prone to asking questions that either a) you don't want asked or b) you don't actually know how to answer because you've never thought about it before. The most common complaint I heard about this supra-agile team was their lack of reported bugs. They'd managed to achieve something like a 98% automation rate for their acceptance and verification testing, freeing up their lone tester to concentrate on sapient activities almost exclusively. Developers were actually excited about writing test cases and pairing with test to improve their coverage. They'd successfully implemented a true continuous integration process where every change was deployed and smoke tested, resulting in a series of builds that were truly customer deployable products.

Everyone acknowledged these successes, but there was still this bug thing. It really bothered people. Project managers complained to me about it. The director of engineering complained to me about it. Other project teams complained to me about it, for some reason. It even bothered me more than a little, and not just because people were coming up to my desk to complain for ten minutes at a go. That's when I approached Max about it, and it's also when he asked me one of his patented questions buried inside the following statement:

"We fix our bugs as soon as they're caught, and then we add a test case to make sure they don't recur. If we do that why would we need to write them down in the bug tracking system?"

Yeah. That was my reaction as well. I'm guessing you sort of thought to yourself, "Yeah, but … but what about… " and then you sort of petered out. I did answer his question finally, but it forced me to re-examine my first principles with regards to software testing and these things we call "bugs." So to answer the question of why we manage bugs in a manner that isn't self-perpetuating (i.e.: We manage bugs so we can keep track of how many bugs we're managing) we need to first define just what a bug is within the scope of a project.

What is a bug?

My article is hopefully not about the different ways we can define a "software bug" since I believe my thesis is applicable no matter which of the competing definitions is used. To support that view, I will now use a definition with which I don't happen to agree but around which there is some agreement and support in different circles.

The IEEE (which people who follow my blog or have read my other work will tell you I almost never cite) defines a bug as,

> *"(1) The difference between a computed, observed, or measured value or condition and the true, specified, or theoretically correct value or condition. For example, a difference of 30 meters between a computed result and the correct result."* [3]

In the interest of making that whole thing easily ingestible, I'll summarize for you all. A "bug" is something you experience that doesn't line up with what you expected to experience based on what you know when it happened. The whole process of reviewing a bug is based on that bit about "what you know when it happened." If you don't know a toss about the design and the expectations or requirements, you're liable to find quite a few things you think are bugs that are really just your lack of knowledge. As you learn more the number of relevant bugs should increase. Consequentially the more the product or system (or more accurately the people designing said system) learns about the expectations and deviations the number of bugs experiences should decrease. Of course no one ever comes at a project as a truly blank slate. There will always be expectations of behavior, even at the code level. We have certain expectations of a credit card verification web service that differ from a desktop calculator application that differ from a customer retention management solution. When we say "we've found a bug" what we really mean is "we've found a spot where what we wanted to implement is not what we actually implemented." Sometimes this results in code that does not even compile. Sometimes this results in a background that's just the wrong shade of blue. "Finding bugs" is then walking through what we actually built, delivered, or installed to see if it's really what we expected it to be in our heads.

At its core then, testing is about learning. The users learn about the system and the system learns about the users. Bugs are one of the primary methods for communication and interaction during that learning process. It should be stressed, however, that bugs are not the only method facilitating this communication. Max's challenge to me was based on this understanding. He informed me the team communicated during their learning process using other methods, namely physical meetings and documented test cases. He challenged me to justify adding another non-product generating activity onto their task load. I eventually got back to him with an answer he couldn't refute even taking into account the teams were supposedly "self-managing" by this point.

What did you eventually say to him?

I'm getting to that, just bear with me a little longer. This prior bit about a self-managing team is important. Do you remember when I brought up who was coming to me to kvetch[4] and complain about the offending team not reporting bugs in our fancy bug tracking system? You do? Do you remember who I said was doing the complaining? Most of them were the manager types (as opposed to producer types) I talked about earlier. These people relied on communication and information from Max's team in order to perform their jobs. The producers on Max's team made the things the managers relied on, were affected by, or were generally interested in following. This reliance created a dependence on what Max's team produced, which caused tension when they trended into a different path.

The reason the manager people didn't like this new trend goes back to the nice little bunches of numbers you can easily generate if you plug information into a modern bug/data management system. The raw data entered can be processed, collated, and analyzed in about as many different combinations as the person doing the processing can imagine. The manager sort (again, see my original definition and don't assume I mean an actual manager) were used to creating their own information to suit their specific needs using the advanced analytical tools these systems put at their disposal. Further to the point, no small amount of cost was incurred coming up with forms and workflows inserted into the tool(s) in order to facilitate this analysis work by standardizing the data input by the producers. The manager types were used to getting their own information based on these tools. They'd grown accustomed to it. They'd all agreed a set of numbers trending in one direction meant one thing, while another meant something else. The producers could be left to producing, leaving off the task of actually communicating information.

Whether or not these numbers and analyses meant what the general wisdom presupposed is not relevant to the actual complaints. What is relevant is the expectation of those data points being entered in a particular form, namely bugs.

The team before the advent of Max had agreed to use bugs as a primary source of data regarding the product health as it worked its way through our development workflows. Since bug data had been collected for several releases prior, there was an assumption of historical relevance. Since similar data had been and was being collected by other product teams; there was an assumption of comparative relevance. By switching the team from one statistical metric (bugs) to another (tests), Max had radically changed all that even though he maintained the validity of statistical analysis for product health. It seemed as though Max was forcing a decision to be made with regards to whether the company was really going to allow the teams to be self-managing or not.

This is Sparta!

Self-managing teams are different. When a company commits to "empowering" teams with "self-management," it has real repercussions on the managers if the commitment is more than just putting lipstick on a pig.[5]

A self-managing team is empowered to develop independent processes and measures which is sort of where the "self" part comes into play. The team decides what are appropriate, accurate, and measurable metrics against which they gauge their success. Information and not data is communicated up the chain to those who must then find a way to do their jobs. If the higher links want raw data in the form of statistics, the onus is on them to somehow contextualize and weight the data to get what they need. By the same token the team must commit to generating the information (not data) the stake holders are requiring.

The project managers and technology leads and IT director in coming to me were in effect asking me to subvert this process. These people were asking me to enforce top down management by telling this team how to deliver data in the form of bugs that could then analyzed in order to extract information. The team was telling me they would deliver data in the form of tests that could be analyzed in order to extract information. Both parties had lined themselves up against each other, shields bristling with spear tips. On the one side of the Thermopylaen conflict was Leonidas and his 300 developers. Against them were arranged the vast hordes of the rest of the company with more than a few allies arraigned behind. Both were claiming their data was correct, and it was on the subject of the data that Max/Leonidas challenged me when I arrived as an envoy to discuss peace. I felt myself standing at the precipice, with Max's foot twitching for a shove.

Of course I wouldn't be writing this article if I didn't come up with a solution, so here it is. To answer all parties and resolve the irresolvable I had to change the conversation. To do this I stopped thinking of these things as data and thought about what information is intrinsic to bugs.

To Infinity and Beyond

We talked earlier about how testing is actually learning. The team learns about the product, their assumptions, and their mistakes. Things like bugs, test cases, meetings, specifications, and the like are all things we create to communicate this learning so we don't repeat the same mistakes. Communicating a bug can take two forms. You can tell someone about it, or you can write it down. You write things down to communicate with someone not right there at that moment. When we do this we usually mean someone not in the room or the building. Of course there's another person to whom we can also write notes that's not in the room right now … our future selves.

Did I just blow your mind? Good.

One of the reasons we write things down is to document and archive our current knowledge. We do this so we're not forced to relearn the same things over and over again. It also theoretically keeps us from making the same mistakes over and over again, but your mileage on that may vary. We write down test cases to document our current expectations, and we write down bugs to document where it was done wrong. The information about deviations and such is vitally important to some projects because people also write down what the "root cause" proved to be and how they "fixed" it eventually.

And that right there, my friends, is information not contained in a test case. It's also the thing I got both parties to acknowledge.

Bugs have a workflow distinct from other artifacts and vice versa, no matter what some integrated tool solution people might claim. Bugs are place holder repository information for learning that contains development and design tasks. In order to understand why changes were made to a solution you need to understand the context. Bugs provide that context, tests do not. A test cases tells you something deviated on one day, but not another. It does not tell you what or why things changed. If a team updates and documents their learning when working on a "bug" they will not lose that learning.

This then, was my answer to both the implied and the actual question. The team needed to enter bugs so they didn't lose the learning they amassed while performing tasks on the project not otherwise documented. If they didn't document this learning, they ran the risk of repeating the same mistakes over and over again as priorities changed and attention was put elsewhere. Out of this statement flowed the rest of the process decisions, and rather rapidly I might add. The team agreed the distributed tool would be good to use because of its rich search features and data management capabilities. The team decided to keep authoring test cases, but to organize them in a slightly different manner now that bugs were being tracked elsewhere. I still had some coaching to do with the manager folk and their concept of "self-managing" but that was a different crisis to address.

If I agree, what's the next step?

This theoretical stuff is all well and good, but what do we do with it as testers who now want to manage our bugs? We take a look at how (and if) we're currently managing them.

We need to revisit the "Five Why" technique[6] to the data we're currently collecting when we enter a bug somewhere to track and manage. We need to ask such questions as "Why are we collecting this data?" rather than just collecting it because the tool asks us to or someone decided it was a "best practice." Ask yourself if you're collecting data on the project component area to aid in sorting and searching if you need to find out information quickly, or if you're using it to judge how "healthy" that component area is when making resource and release determinations. If it's not helping you search for things, try getting rid of it and seeing if there's another way you can communicate product health and fitness for release.

I hear test cases are an excellent measure of such things, as a suggestion.

Secondly you should make sure all data you need for historical communication is actually being entered. A very commonly overlooked or simply ignored piece of historical record is the conversation that occurred between test and development when triaging and debugging the bug. On distributed or even compartmentalized teams this conversation often occurs in either email or in instant messaging sessions. Unfortunately those valuable records are lost when they are not included in the bug itself. Another valuable record not as commonly left out but is still something to monitor is the resolution strategy. How was the bug actually fixed? All too often I've delved into a bug system and come up with what looks like an exact match for what we're seeing right at this very moment in production or in our test environment. Unfortunately the only record kept about the actual root cause is often a note from the developer signifying the bug is fixed, which build it's expected to be included within, and a note from the tester verifying it's no longer reproducible. The developer and tester must then wade through the source code management system, looking for code changes deployed with that build (or possibly one around it) to determine what might have been changed. From that best guess, the team then spends time attempting to reconstruct what might have been a problem causing the change to be made. It's a lot of expensive work that could have been prevented if the developer had simply spent two extra minutes elaborating on "fixed" and the tester had elaborated on "not reproducible in current build."

So moving forward with the theory, make it useful. Make sure you're collecting what you need from bugs that make bug tracking useful. If you don't care about that stuff, then take Max's route and try collecting and analyzing something else. You may find it far superior for your actual goals in the long run.

Notes:

[1] See 1999's "Blade" directed Stephen Norrington and starring Wesley Snipes. There is no correlation here. It's just a good movie I've always wanted to reference in footnotes.

[2] If you'd like to read articles, classes, and books by Max, I recommend checking out his company Hexagon Software at http://www.hexsw.com/ and bring your mouth guard.

[3] 610.12-1990 - IEEE Standard Glossary of Software Engineering Terminology, http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=159342

[4] It's a Yiddish word. It sort of means to complain or to whine about something, I suppose. http://en.wiktionary.org/wiki/kvetch

[5] A popular cliché from the United States referring to …you know what? Just Google these from now on.

[6] For more information on this technique, see http://en.wikipedia.org/wiki/5_Whys

Biography

Curtis Stuehrenberg is a recent transplant to the San Francisco Bay Area with over a decade pursuing a career where he gets to break things. He's the founder and chief officer of his company Cowboy Testing as well as the author of the popular **WordPress blog** of the same name.

Currently he is helping the leading manufacturer of scientific informatics software for chemical research, Accelrys INC, as an advisory software test engineer tasked with guiding them through the always painful transition to agility. Somehow he's also been able to squeeze in time to squeeze in teaching classes on exploratory testing, moderating popular software quality assurance professional forums, and working with people like **Michael Larsen** to kick start a Bay Area Software Testing group.

Curtis can be contacted on Twitter @cowboytesting or on cstuehrenberg@gmail.com

Discuss this topic on Quality Testing
Click HERE

Back To Index

Testing... with Anurag

Test Battery & Power consumption of your mobile app with Nokia Energy Profiler

Friends, in my last article we discussed about some points which can help you "Getting Started with Mobile Application Testing". As I discussed earlier, it is always advisable to use some tools for monitoring the behavior of application under test. Today we are going to talk about an utility/application for Symbian platform (S60 3rd Edition, Feature Pack 1 and later devices.), which will help you to test and monitor the applications energy usage in real time on target devices. Please note that while testing your mobile application you should always keep your eye on the power consumption of the device while using your application. Make sure that you application is not making the device to drain too much battery/power.

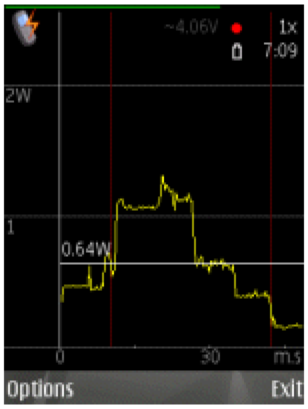
Biography

Anurag Khode is a Passionate Mobile Application test engineer working for Mobile Apps Quality since more than 4 years.

He is the writer of the famous blog **Mobile Application Testing** and founder of dedicated mobile software testing community **Mobile QA Zone.**

His work in Mobile Application testing has been well appreciated by **Software testing professionals** and **UTI** (Unified Testing Initiative, nonprofit organization working for Quality Standards for Mobile Application with members as Nokia, Oracle, Orange, AT & T, LG Samsung, and Motorola). Having started with this column he is also a Core Team Member of **Tea-time with Testers**. Contact Anurag at **anurag.khode@hotmail.com**

What is Nokia Energy Profiler?



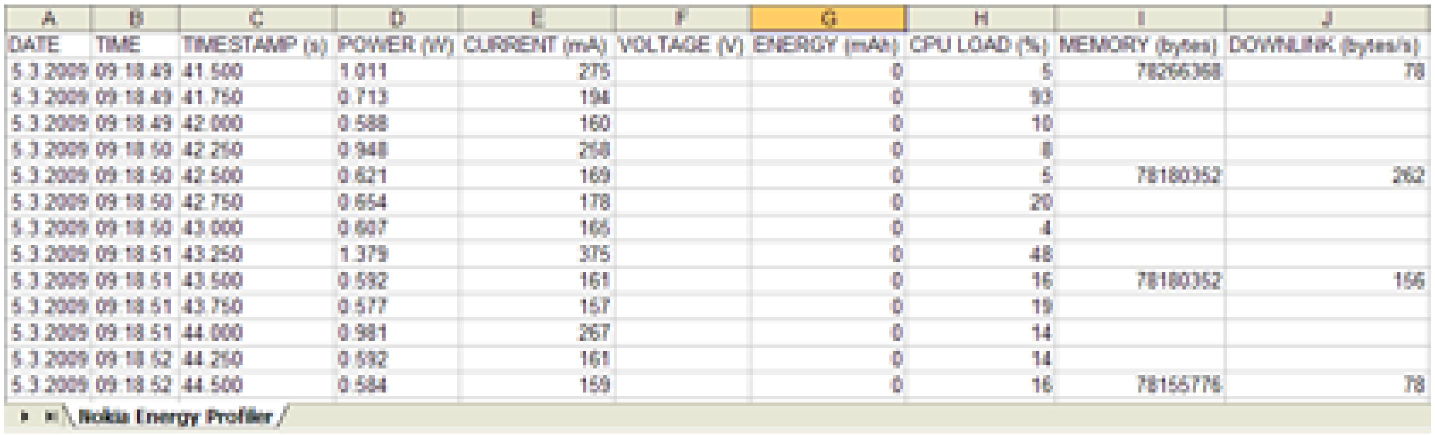
Nokia Energy profiler is a standalone test and measurement application which lets you monitor the battery consumption on target device. This application will help you monitor the power consumption ,Check your internet connection speed., investigate cellular network's coverage, watch the processor load during applications ,RAM usage and much more.

For which mobile devices is Nokia Energy Profiler available?

The Nokia Energy Profiler's measurement features work only on S60 3rd Edition, Feature Pack 1 and later devices.

How to use Nokia Energy Profiler for Testing?

- Run Nokia Energy Profiler (Keep this app running in Background)
- Perform testing on your Target application.
- Once testing is completed, go back to the Nokia Energy Profiler application and stop the measurements
- Save the results
- Study the results or
- Export results as CSV



A	B	C	D	E	F	G	H	I	J
DATE	TIME	TIMESTAMP (s)	POWER (W)	CURRENT (mA)	VOLTAGE (V)	ENERGY (mAh)	CPU LOAD (%)	MEMORY (bytes)	DOWNLINK (bytes/s)
5.3.2009	09:18:49	41.500	1.011	275		0	5	78266368	78
5.3.2009	09:18:49	41.750	0.713	194		0	93		
5.3.2009	09:18:49	42.000	0.588	160		0	10		
5.3.2009	09:18:50	42.250	0.948	258		0	8		
5.3.2009	09:18:50	42.500	0.621	169		0	5	78180352	262
5.3.2009	09:18:50	42.750	0.654	178		0	20		
5.3.2009	09:18:50	43.000	0.607	165		0	4		
5.3.2009	09:18:51	43.250	1.379	375		0	48		
5.3.2009	09:18:51	43.500	0.592	161		0	16	78180352	156
5.3.2009	09:18:51	43.750	0.577	157		0	19		
5.3.2009	09:18:51	44.000	0.981	267		0	14		
5.3.2009	09:18:52	44.250	0.592	161		0	14		
5.3.2009	09:18:52	44.500	0.584	159		0	16	78155776	78

Nokia Energy Profiler

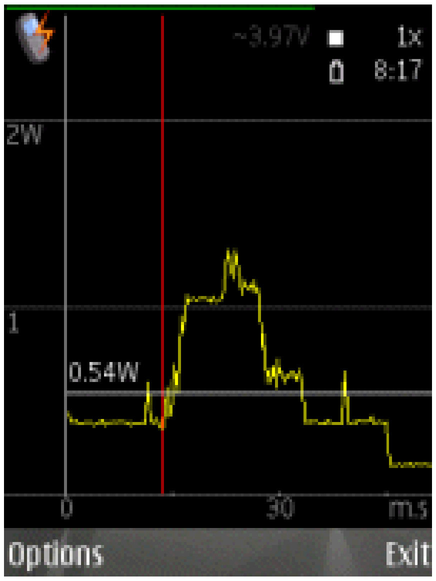
What can I do with Nokia Energy Profiler?

The Nokia Energy Profiler supports the following views:

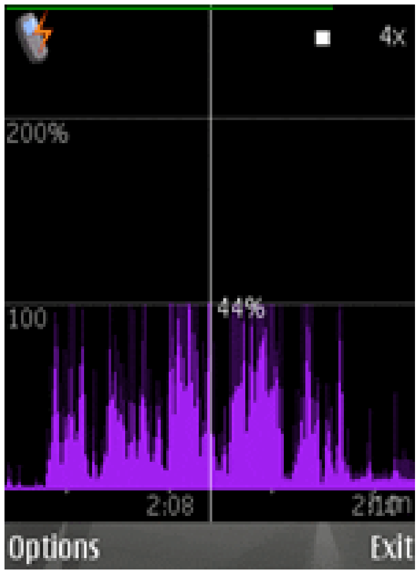
- Power View
- Current.
- Processor.
- RAM Memory.
- Network Speed.
- WLAN.
- Signal Levels.
- Energy.
- Voltage.
- 3G Timers.

Power View:

Power view shows power consumption over a measurement period. The battery time (corner indicator) gives you the estimated time that will elapse before the battery is fully discharged. Check out figures:



Power View

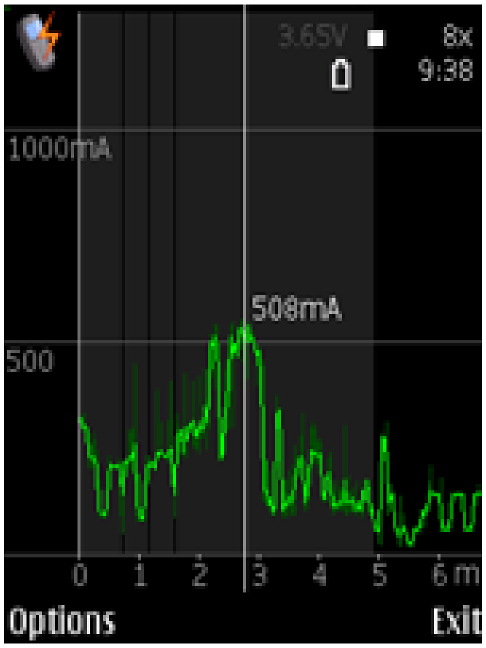


Processor View

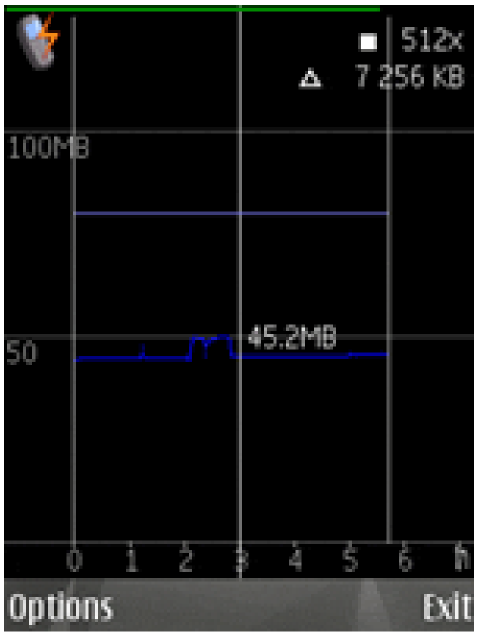
Processor View:

The Processor view gives you idea about CPU-processing capability. 0 Percent should be considered as no processing where as 100% indicates maximum processing. (Please note that different mobile devices have different CPUs, Cache sizes, memory bandwidth and chipset performance.) See figure above.

Current View: Current view displays current consumption. As the battery discharges, voltage drops and current increases so that power consumption stays roughly constant.



Current View



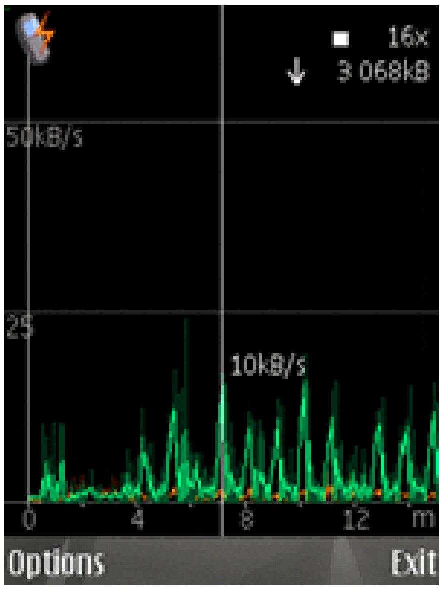
RAM Memory View

Ram Memory View:

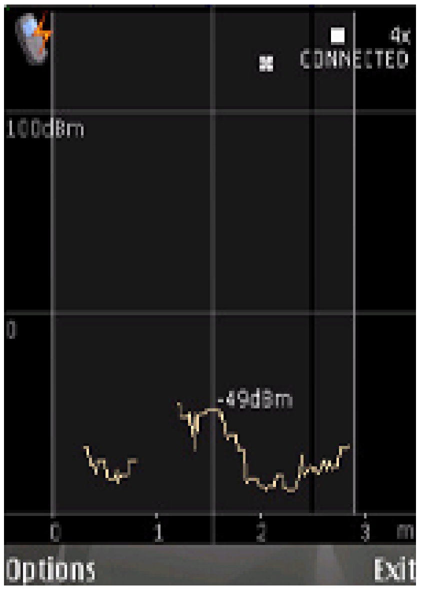
RAM Memory view displays the RAM memory usage over the period. The unit is megabytes. Check figure on last page.

Network View:

Network view presents the downlink (download) and uplink (upload) speeds through the IP stack.



Network View



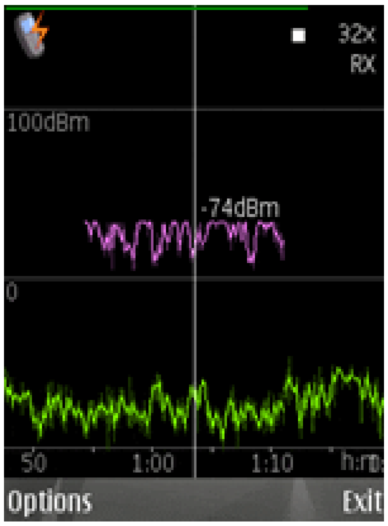
WLAN View

WLAN view:

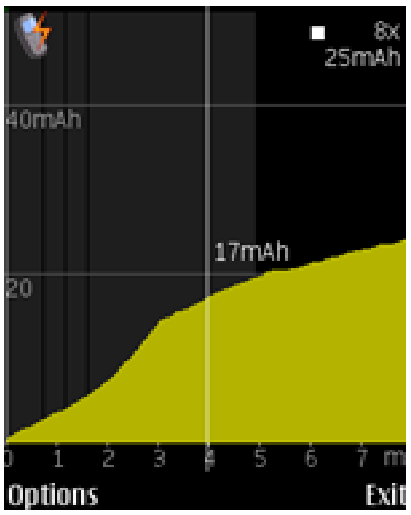
WLAN view enables you to study application behavior in various signal-strength conditions.

Signal Levels view:

Signal Levels view shows the cellular signal levels as RX and TX levels. The RX level corresponds to the power of the received cellular signal.



Signal Levels View



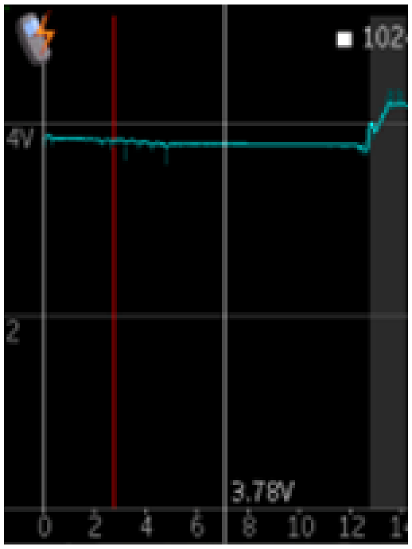
Energy View

Energy view:

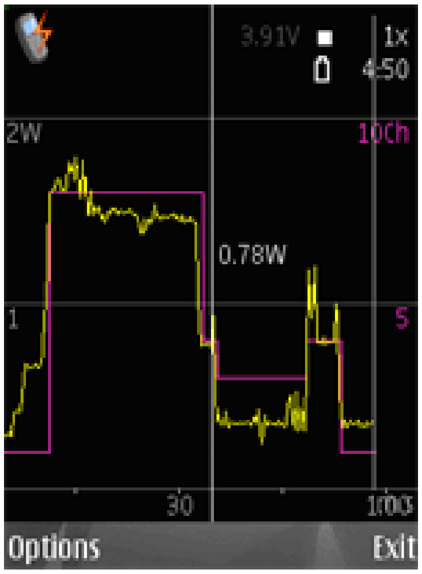
Energy view shows the cumulative energy consumed over the measurement period. Figure on last page.

Voltage View:

Battery voltage can be examined in Voltage view.



Voltage View



3G Timers View

3G Timers view:

3G Timers view shows the 3G-network-data inactivity timers, also known as T1 and T2.

From where can I get more details about Nokia Energy Profiler?

You can get further details about Nokia Energy Profiler Installation, Usage, Features here .

From where can I download Nokia Energy Profiler?

You can download Nokia Energy Profiler from here .

Do not forget to follow me on twitter-@anuraagrules, @mobileapptest, @mobileqazone or drop me a mail at anurag.khode@hotmail.com .

Back To Index

testing
intelligence

- its all about becoming an intelligent tester

an exclusive series by **Joel Montvelisky**

5 Ideas on how "User Profiles" can Improve Your Testing!

As testers we always look for ways to test better, to find the critical bugs first and to focus on the most important areas of the application under test; after all we have limited time to test and we need to make the most out of it.

One of the most important tools a tester has in his "virtual toolbox" is the use of *User Profiles*.

What is a User Profile?

A User Profile is the representation of a generic user of your system. It does not relate to any specific user but to the way you can describe the average user(/s) of your application.

In practice, most real applications have a number of relevant User Profiles such as regular user, managers, system administrators, external users, etc.; and for each of them you will want to create a separate Profile.

When creating a User Profile you will define all the personal and professional traits that are relevant to the work this person does with your application.

For example:

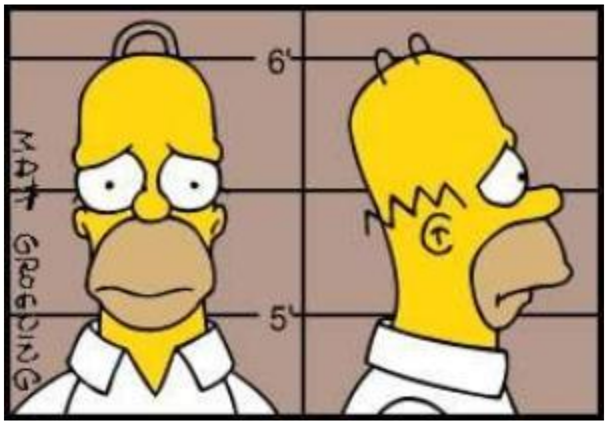
- Years of experience in the field

- Knowledge of your tool, or similar tools

- The way in which he/she works: on a desk in his office, in the field while walking, on a store behind a counter, does the person access the application from his iPhone or BlackBerry?

- Interactions with other users of your tool or other tools with which they need to communicate and collaborate

- You may even want to define some demographic information if relevant such as age, nationality, language, etc.



Some companies (and I really like that approach) go as far as creating big posters with the description of their User Profiles, including their Name and even a Picture. Even though this may sound unimportant and even childish it will help you to relate to your User Profiles and to remember who is who better and faster.

5 simple and powerful things you can do with User Profiles:

(1) Write Testing Requirements and Documentation with User Profiles

User Profiles as part of your requirement and design documents allow you to create more concrete scenarios of what the application should do based on the needs and characteristics of concrete individuals, this in turn will allow you as a tester to find problems faster during your learning and review process.

By providing feedback and comments based on User Profiles you also help others to understand you better and to agree or disagree with you based not on their personal assumptions but on what was already agreed upon by the whole team.

(2) Test Scenarios based on User Profiles

One of the most powerful things you can do with User Profiles is to leverage them when creating your testing scenarios.

Profiles will allow you to work always based on realistic scenarios, allowing you to test the things that will surely happen in the outside world once you release your product.

You should use Profiles not only to create your positive scenarios but also to think about the wrong things or negative scenarios that your users run into based on their constraints and the ways they work with your product.

(3) Test Prioritization based on Profiles

Once your tests take into account User Profiles it will be easier for you to prioritize them based on their importance, risk and even on how common the scenario is and how many user will fall into a bug located on this area.

You obviously want to run all our tests, but just in case you start running out of time this is an additional way to know what not to test…

(4) Calculate coverage based on User Profiles

User Profiles are a simple and more effective way to communicate your testing coverage to external stakeholders. For example when you talk to your Marketing or Sales guys, it will make more sense to them if you talk about what you've tested and haven't tested based on User Profiles; it will allow them to understand the implications and the risks faster and easier.

(5) Work with realistic test data based on your User Profiles

Profiles can be used to generate more realistic data and not to use sentences and phrases such as "this is a test" and "hello world", and data entries such as "asdf' or "123456". When you work with realistic and varied data you have a better chance of running into important bugs that will happen in the field.

In PractiTest we constantly leverage "User Profiles" as one of our main sources for testing. They've proven to be effective and to provide great value for the limited amount of effort needed to create them.

Try them out and let me know how they worked for you.

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://qablog.practitest.com and regularly tweets as joelmonte

This is a very good initiative in the field of Software testing. The magazine contents are really helpful as it covers most of the recent updates in Software testing field.

Thank you. Looking forward for the answers and more crossword / puzzles...which are also interesting ;-) - Jesal

Call for Articles !

Have you got something to say?

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

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

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

Submit your articles, stories, thoughts around software testing.

now its your chance to be heard...!

sciencephotogallery

How many hairs do you have on your head?

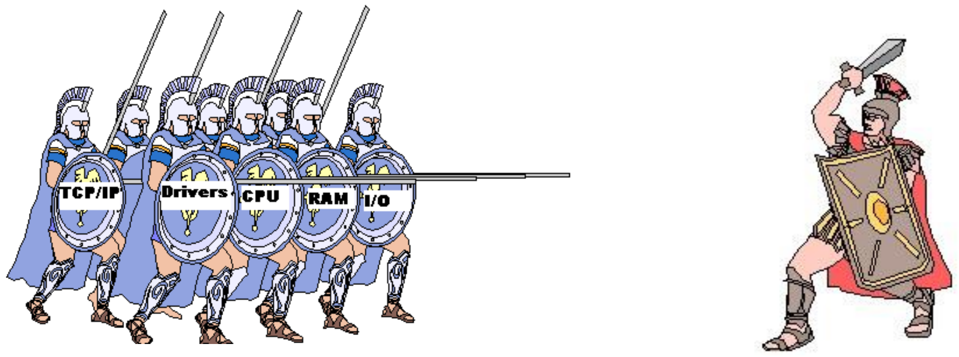
One of my favourite questions that I ask when I mentor testers at companies is "How many hairs do you have on your head?" I'm always amazed at the answers that I get from the participants. Some give me a number of 10,000 while some enthusiastic participants give me a number of ONE million. Isn't it amazing that we have of variance of 1000x for such a simple question, the one that we are probably very familiar with.

It is interesting to note that the participant profile typically consists of a mix of maturities ranging from senior and mid-level engineers. Nonetheless the answers that I get from senior participants are equally crazy. Why is there such a large variation?

My hypothesis is that engineering staff find it difficult to deal with uncertainty, those aspects whose "countability" is fuzzy. So the typical answers seem to be based on seat-of-the-pants approach. In the few cases where we've encountered a similar situation and solved it well, the estimates turn out to be correct; otherwise it's simply a shot in the dark, praying to God that it turns out to be correct.

It is important to understand that an exact value may not be required; rather it is the reasoning that allows us to come up with a value, and constantly improving the reasoning to better the value. The simplest way to answer the "Hair question" is to compute this by multiplying the density of hair and area of head approximated as square. The answer can be refined by improving the reasoning, that the hair density varies across the head and that we may need to consider the curvature factor of head, hair shed rate etc.

So what are we doing? Rather than just guess somehow, we have tried to construct a simple formula that is continually refined to better the outcome. Note that what we have done is the age old problem solving technique–"divide and conquer".



The act of approximation is very natural; it is in fact part of our instinct. Think about this - we do not take measured steps when we walk, we do not calculate the exact distance when reversing the car and so on. Our natural learning system continuously learns in the background and constantly adjusts the variables to refine the approximation.

Let me illustrate how "scientific approximations" is useful in testing...

One of the common questions that gets asked frequently is "How much time/effort will it take to test this?" A correct answer is elusive. A group of people believe that this can be answered only by people with rich experience, whilst some folks in the community believe this question is stupid as this cannot be estimated. However considerable effort has been expended in building empirical models that would magically answer this question, but they still seem to fall short. What is important here is not the exact answer, rather it is the reasoning that enables us to scientifically approximate. Rather than attempting to answer such questions in a simple binary fashion, a reasoning that identifies the variables involved and connecting them by a formula, that is continually refined allows us to find the right answer. Some of the variables may be #scenarios, #cycles, times involved in understanding, design, automation, execution, mode of execution, volatility of a feature, types of tests etc.

I have observed that when people design load, stress, performance, scalability and volume tests, the test data values used for load profile, size of data are seemingly good guesses. Scientific approximation to estimate the load could be based on these variables- types of end users, #users per user type, features used by an end-user, frequency of their usage, seasonal variations, rate of arrival, growth of business and then connecting these by formula. Continuing on the same train of thought, the data size (volume) can be estimated by identifying the data generating transactions, size of data generated per transaction, the retention period(after computing the number of transactions).

As a professional tester, we encounter a variety of situations that require us to come up with a value/number. Just because we don't see a direct connect to this value does not mean that we resort to "guesswork". What is needed is scientific approximation, the keyword is **scientific**.

A few years ago I read a wonderful book titled "The Art of Profitability" by Adrian Slywotzky.

This book deals with "how to deliver higher profits", where David Zhao an extraordinary teacher beautifully breaks down the problem as a collection of various profitability models and teaches this to his protégé Steve Gardner, a CEO of an ailing multi-billion Dollar company. The teacher uses a Socratic style of teaching approach to enable the CEO to discover reasoning. In one of the chapters, when the teacher asks the CEO for profitability percentages for a product line, the CEO does not have the answer handy. This is when the teacher explains the importance of approximations. In fact he asks the CEO "How long would it take to cart Mount Fuji away with dump trucks, one truckload at a time?" and expects the answer in a minute.

This book inspired me to develop the "Approximation Principle" that is a core concept in HBT (Hypothesis Based Testing).

So the next time, when you encounter a curved ball- a question that does not seem to have clear answer in terms of "a value", don't react immediately. Pause, reason, identify the various variables, connect them via a simple formula, compute the first value, test the value quickly and refine. Sounds involved and time consuming? Try it once and you will be surprised at how quickly this can be done.
I have tried it a number of times and it has been fun.

Want to try it now? You have been doing a great job of testing applications/products for your company. How much money do you think you have saved for your company because of your good testing? Sounds like a good ploy to demand raise!

Stop counting your hair now and start thinking!

Cheers! Enjoy the fuzziness.

If you liked, tweet! My Twitter ID is ash_thiru.

Discuss this topic on QualityTesting
Click HERE

Biography

stag

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

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

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

Back To Index

Quality Testing

Quality Testing is a leading social network and resource center for Software Testing Community in the world, since April 2008. QT provides a simple web platform which addresses all the necessities of today's Software Quality beginners, professionals, experts and a diversified portal powered by Forums, Blogs, Groups, Job Search, Videos, Events, News, and Photos.

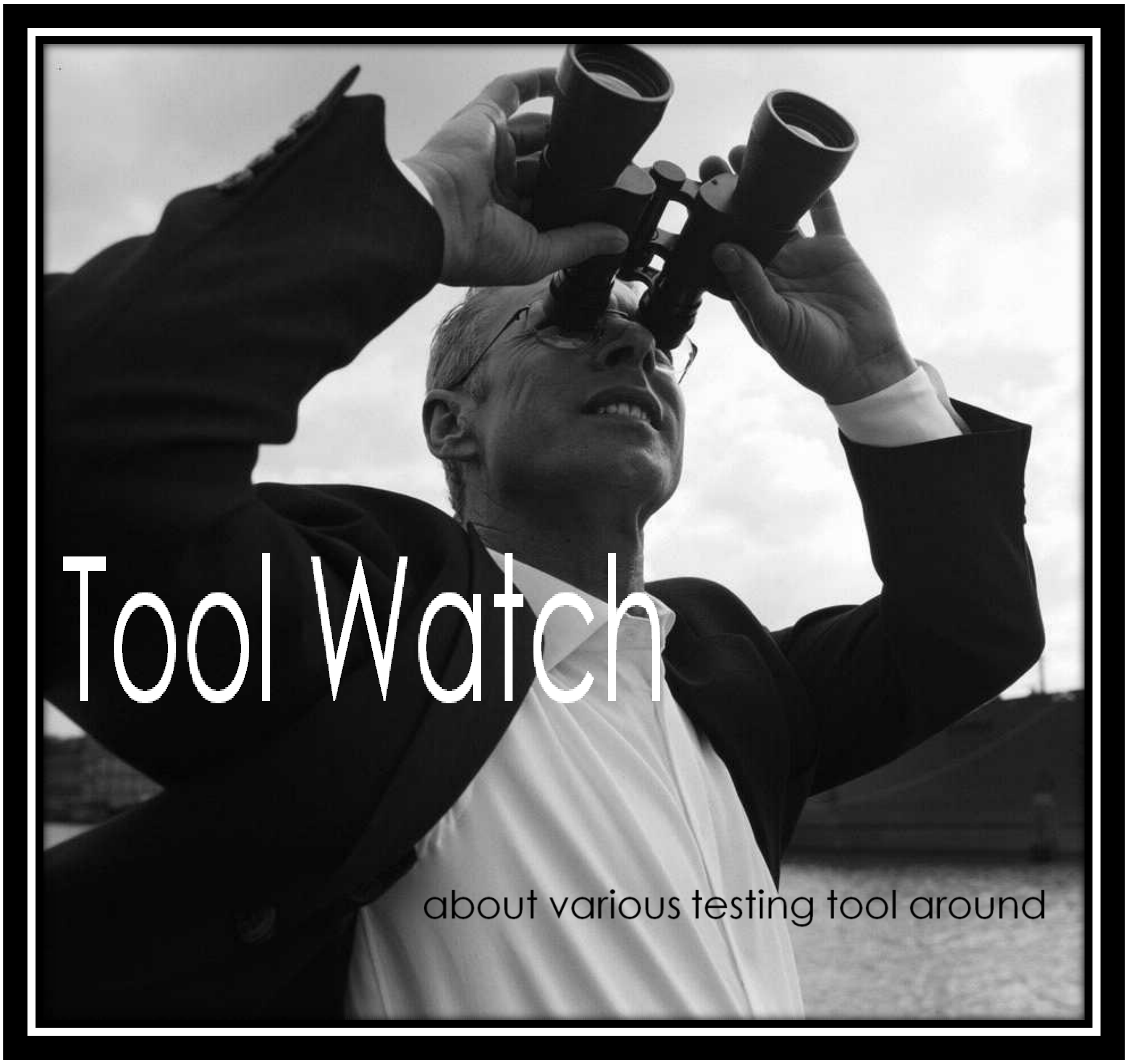
Quality Testing also provides daily Polls and sample tests for certification exams, to make tester to think, practice and get appropriate aid.

Mobile QA Zone

Mobile QA Zone is a first professional Network exclusively for Mobile and Tablets apps Testing.

Looking at the scope and future of mobile apps, Mobiles, Smartphones and even Tablets , Mobile QA Zone has been emerging as a Next generation software testing community for all QA Professionals. The community focuses on testing of mobile apps on Android, iPhone, RIM (Blackberry), BREW, Symbian and other mobile platforms.

On Mobile QA Zone you can share your knowledge via blog posts, Forums, Groups, Videos, Notes and so on.


Tool Watch
about various testing tool around

StressTester™

PART 2

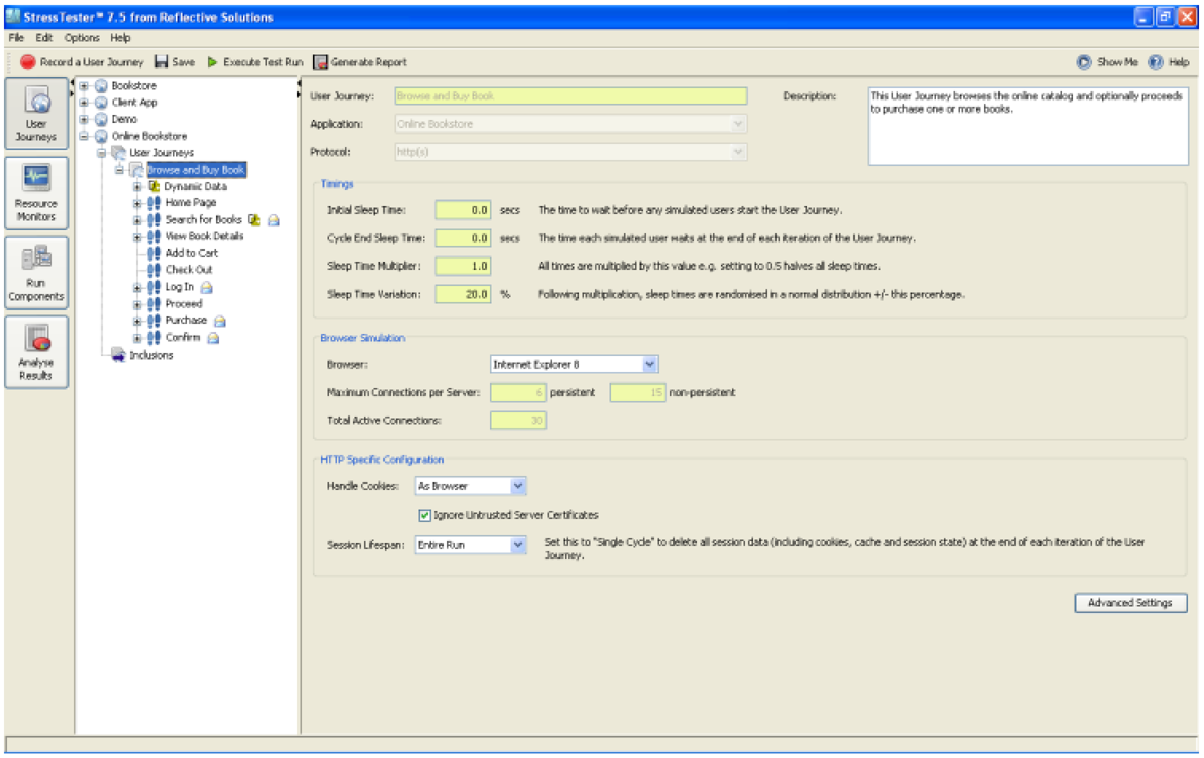
Tea-time with Testers Rating: ★★★★☆

Download our July issue for Part 1

by Juhi Verma & Sharmistha Priyadarshini

Inspecting Your User Journey

Your newly recorded User Journey will be automatically selected within the User Journey workspace and should look something similar to the User Journey displayed below.



User Journey: Browse and Buy Book

Application: Online Bookstore

Protocol: http(s)

Description: This User Journey browses the online catalog and optionally proceeds to purchase one or more books.

Timings

Initial Sleep Time: 0.0 secs The time to wait before any simulated users start the User Journey.

Cycle End Sleep Time: 0.0 secs The time each simulated user waits at the end of each iteration of the User Journey.

Sleep Time Multiplier: 1.0 All times are multiplied by this value e.g. setting to 0.5 halves all sleep times.

Sleep Time Variation: 20.0 % Following multiplication, sleep times are randomised in a normal distribution +/- this percentage.

Browser Simulation

Browser: Internet Explorer 8

Maximum Connections per Server: 6 persistent 15 non-persistent

Total Active Connections: 30

HTTP Specific Configuration

Handle Cookies: As Browser

[✓] Ignore Untrusted Server Certificates

Session Lifespan: Entire Run Set this to "Single Cycle" to delete all session data (including cookies, cache and session state) at the end of each iteration of the User Journey.

Step Groups and Steps

The icon indicates a Step Group and you can expand the Step Group node to see the individual steps within the Group. For web applications the Step Group will be the URL of the web page, and the steps within the Group will be the images, cascading style sheets, etc., that make up the web page.

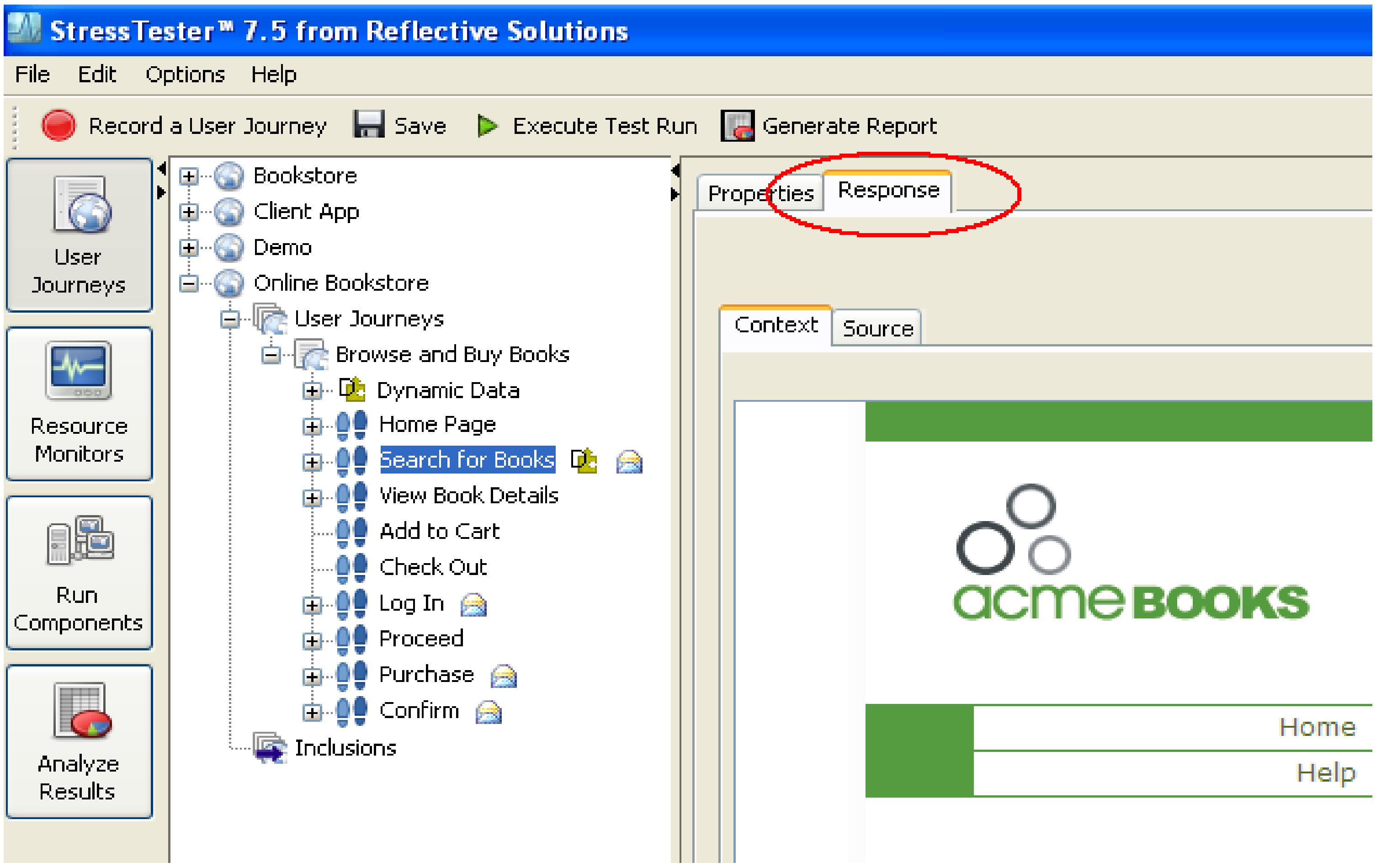
Viewing Responses

At the top of the displayed step properties screen, you will see two tabs: Properties and Response. If you select the Response tab, you can see the response that was returned by the application.

If the response is a visual element or a part or full web page, it will be rendered.

If the response is not displayable you will be able to see the source of the response by clicking the Source tab on the screen.

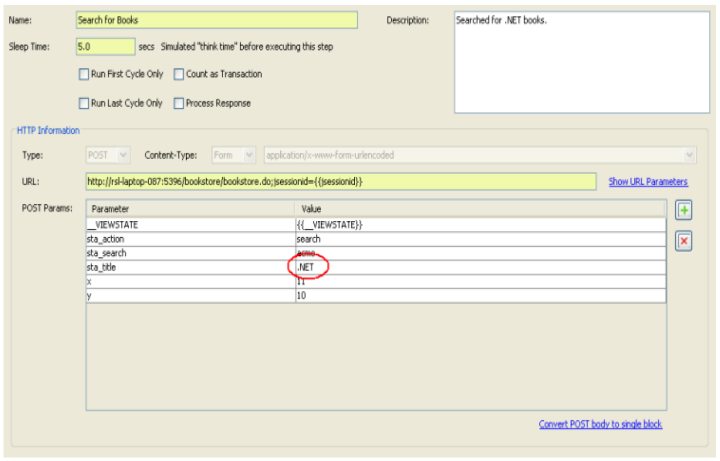
The example below shows the response for the "Search for Books" Step Group of the recorded User Journey.



Request Parameters

Presuming you planned your recording as suggested; your User Journey should contain steps where you entered data into fields in the application. Find one of these steps (you can always right-click on the User Journey node and select 'Search/Replace') and select its node in the navigation panel. You will see the step's properties displayed and should be able to find the data you entered either in the URL, a URL parameter, a POST parameter or in the POST data block.

In the example below, the "Search for Books" step contains the data that was entered during the recording (".NET") as a POST parameter.



Name: Search for Books
Description: Searched for .NET books.
Sleep Time: 5.0 secs Simulated "think time" before executing this step
☐ Run First Cycle Only ☐ Count as Transaction
☐ Run Last Cycle Only ☐ Process Response

HTTP Information
Type: POST Content-Type: Form application/x-www-form-urlencoded
URL: http://rsl-laptop-087:5396/bookstore/bookstore.do;jsessionid={{jsessionid}} Show URL Parameters
POST Params:

Parameter	Value
__VIEWSTATE	{{__VIEWSTATE}}
sta_action	search
sta_search	acme
sta_title	.NET
x	11
y	10

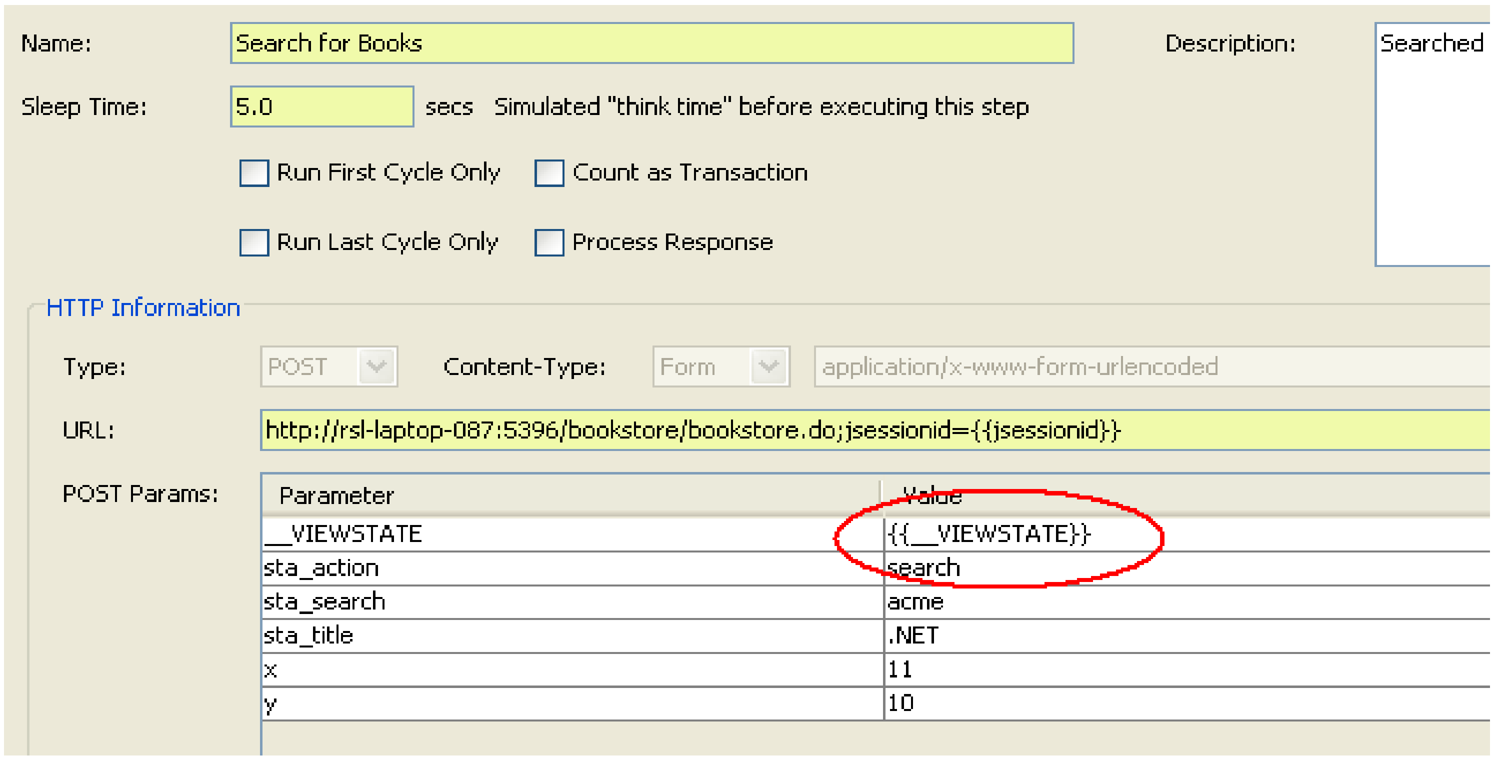
Convert POST body to single block

Session Data

If the post recording wizard detected session variables and you selected the default action of StressTester handling these automatically, you will probably see one or more steps (you may need to expand the Step Groups to find them) that have the icon after the step name in the navigation tree.

This icon identifies that the step contains Dynamic Data – StressTester terminology for values that are varied at run time (parameters).

In the example User Journey shown previously, the "Search for Books" step is followed by the Dynamic Data icon and it can be seen below that the step contains StressTester's automatic parameter to handle .NET's __VIEWSTATE variable.



Name: Search for Books
Description: Searched
Sleep Time: 5.0 secs Simulated "think time" before executing this step
☐ Run First Cycle Only ☐ Count as Transaction
☐ Run Last Cycle Only ☐ Process Response

HTTP Information
Type: POST Content-Type: Form application/x-www-form-urlencoded
URL: http://rsl-laptop-087:5396/bookstore/bookstore.do;jsessionid={{jsessionid}}
POST Params:

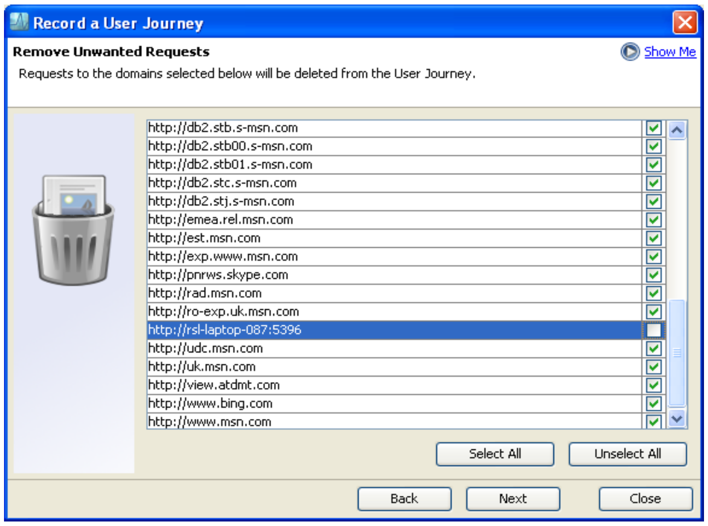
Parameter	Value
__VIEWSTATE	{{__VIEWSTATE}}
sta_action	search
sta_search	acme
sta_title	.NET
x	11
y	10

Post Recording Wizard

When you have recorded your User Journey, you should click the ■ button on the Annotation Window to close the window. The main StressTester window will be displayed and the post recording wizard will start.

Remove Unwanted URLs

If the recording captured requests to more than one domain, all domains will be listed. If you wish to remove all User Journey steps for a particular domain (e.g. remove all the calls to Google Analytics), select the domain(s) to remove.


Record a User Journey

Show Me

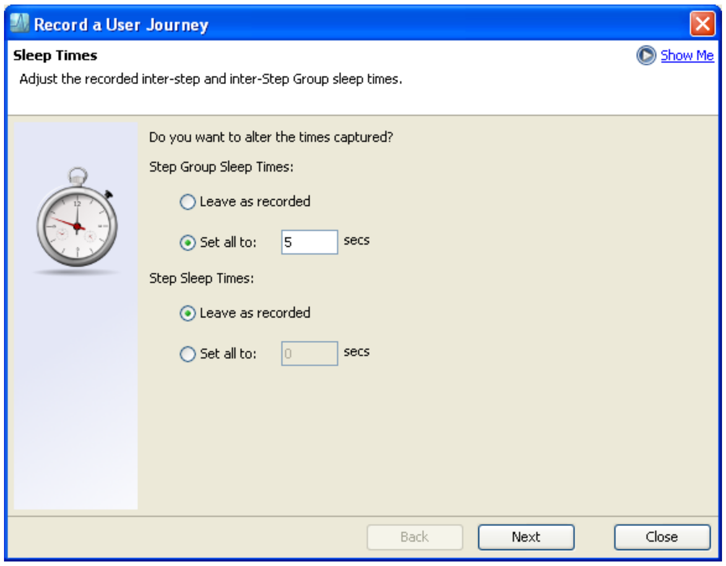
Remove Unwanted Requests

Requests to the domains selected below will be deleted from the User Journey.

http://db2.stb.s-msn.com	✓
http://db2.stb00.s-msn.com	✓
http://db2.stb01.s-msn.com	✓
http://db2.stc.s-msn.com	✓
http://db2.stj.s-msn.com	✓
http://emea.rel.msn.com	✓
http://est.msn.com	✓
http://exp.www.msn.com	✓
http://pnrws.skype.com	✓
http://rad.msn.com	✓
http://ro-exp.uk.msn.com	✓
http://rsl-laptop-087:5396	
http://udc.msn.com	✓
http://uk.msn.com	✓
http://view.atdmt.com	✓
http://www.bing.com	✓
http://www.msn.com	✓

Sleep Times

Clicking 'Next' allows you to set all the sleep time in the User Journey to be the same, either for Step Group lead steps or steps within Step Groups, or leave them set to the values that were recorded.


Record a User Journey

Show Me

Sleep Times

Adjust the recorded inter-step and inter-Step Group sleep times.

Do you want to alter the times captured?

Step Group Sleep Times:

Leave as recorded [✓ Set all to: 5 secs]

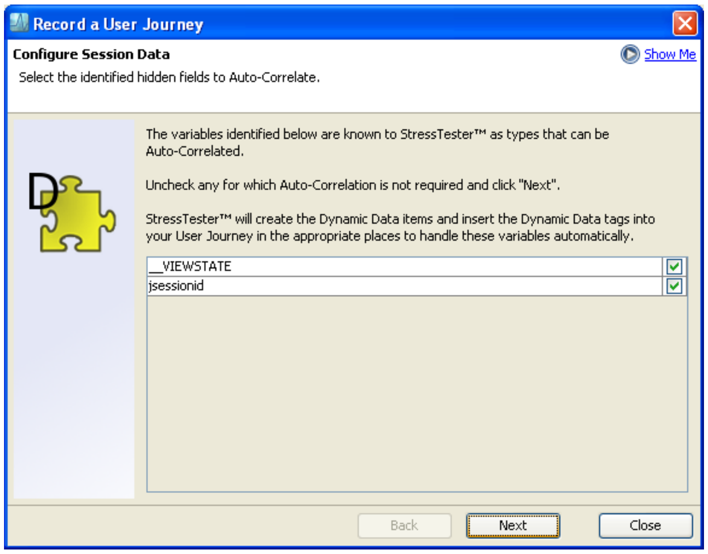
Step Sleep Times:

[✓ Leave as recorded] Set all to: 0 secs

Note :

It is normally advisable to leave the step sleep times to the values that were recorded.

Session Data


Record a User Journey

Configure Session Data

Select the identified hidden fields to Auto-Correlate.

Show Me

The variables identified below are known to StressTester™ as types that can be Auto-Correlated.

Uncheck any for which Auto-Correlation is not required and click "Next".

StressTester™ will create the Dynamic Data items and insert the Dynamic Data tags into your User Journey in the appropriate places to handle these variables automatically.

__VIEWSTATE	✓
jsessionid	✓

Back | Next | Close

It is normal to let StressTester automatically handle any session variables found so you should normally not need to change the default settings (all found variables to be handled automatically).

If your application has its own hidden variables that you would like StressTester to search for and handle automatically.

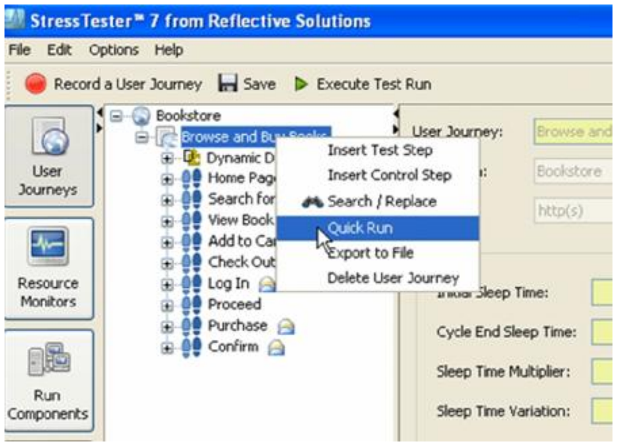
Clicking 'Next' will confirm that the post recording wizard is finished and you should click 'Finish' to close the wizard.

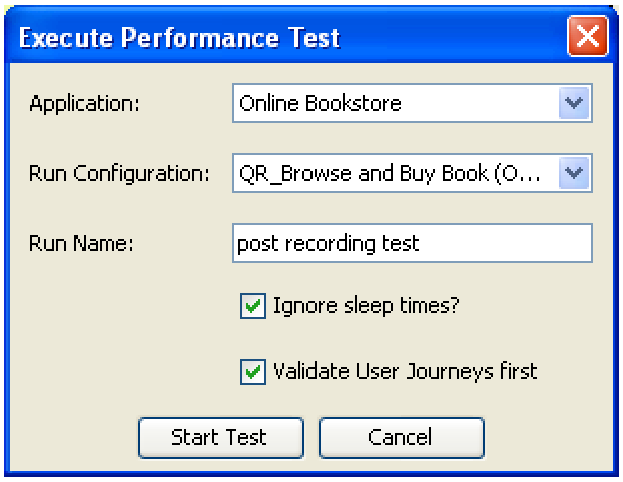
Test Your User Journey

You are now ready to test your User Journey using StressTester's Quick Run functionality.

Execute a Quick Run

To Quick Run your User Journey, right-click on the User Journey node and select 'Quick Run'. This will open the Execute Performance Test dialogue as shown below. Enter a Run Name for the Quick Run test (e.g. "post recording test") and leave the options to their default selections.




Execute Performance Test

Application:	Online Bookstore
Run Configuration:	QR_Browse and Buy Book (O...
Run Name:	post recording test

☑ Ignore sleep times?

☑ Validate User Journeys first

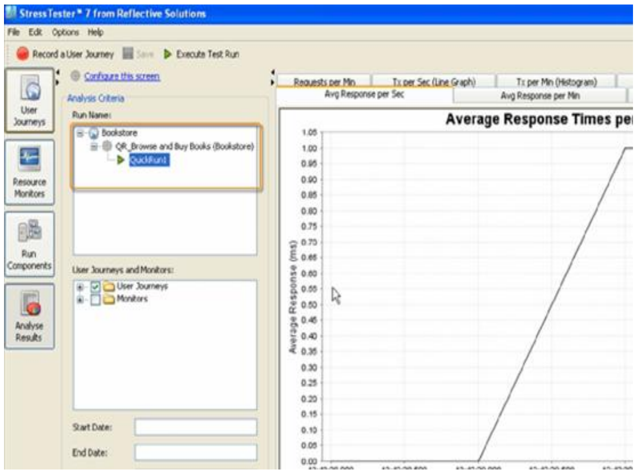
Start Test | Cancel

Clicking 'Start Test' button will switch you to the Analyze Results workspace.

Note :

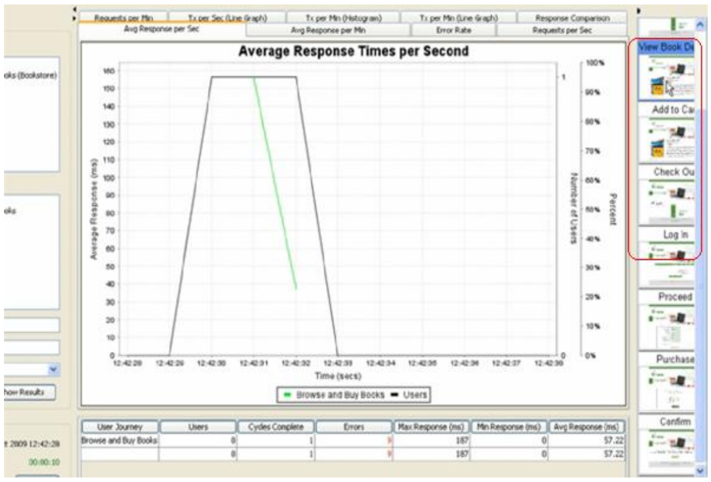
Validate user journey's first option will check that the User Journey is valid before executing the test.

Viewing the Results



Within this workspace the currently executing Quick Run will be selected in the navigation panel.

You will see a graph displaying the average response time of Step Groups and a data table representing the test results.



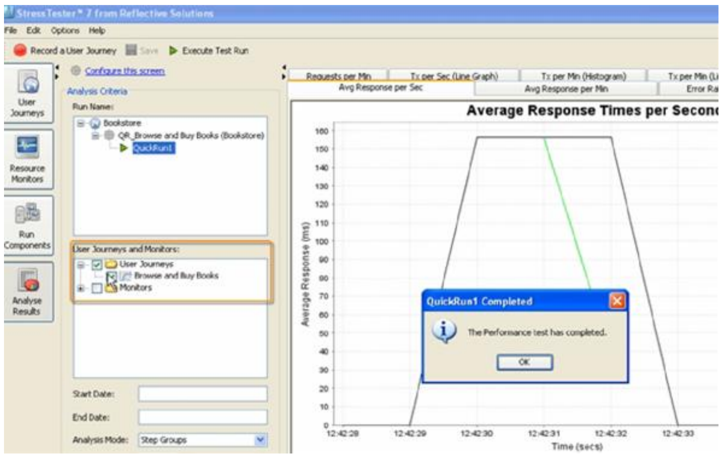
User Journey	Users	Cycles Complete	Errors	Max Response (ms)	Min Response (ms)	Avg Response (ms)
Browse and Buy Books	0	1	0	187	0	57.22
	0	1	0	187	0	57.22

However, what is of most interest during a Quick Run is the thumbnails displayed down the right-hand side of the workspace. Each thumbnail represents the results of one Step Group (web page).

Analyzing Quick Run Results

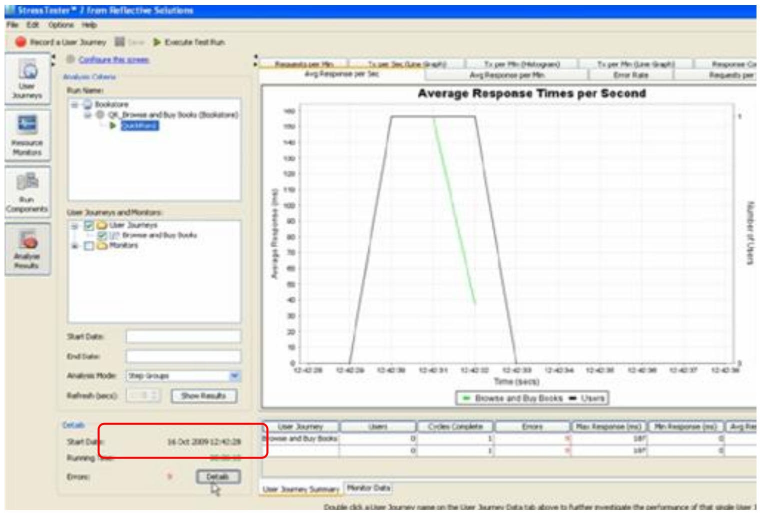
The first thing to look at is the thumbnails displayed down the right-hand side of the workspace. If you double-click on a thumbnail, the response returned by the application will be displayed in your default browser.

You should check that the responses captured in the Quick Run are what you would have expected.


StressTester™ 7 from Reflective Solutions
Average Response Times per Second
Time (secs)
OK

QuickRun1 Completed

The Performance test has completed.



Start Date: 16 Oct 2009 12:42:28

Double click a User Journey name on the User Journey Data tab above to further investigate the performance of that single User J

The second thing to look at is the error counts; both in the data table at the bottom of the workspace and the Errors value at the bottom of the navigation panel.

If either of these indicates errors, you can look at the Error Details and Error Summary detail panel tabs for further information about the errors detected by StressTester.

Diagnosing Quick Run Problems

This section describes some of the common problems you may encounter when attempting to Quick Run a new User Journey.

Missing Web Page Content

If, when looking at the thumbnails or full web page responses, you notice that images or other web page content is missing, this is likely to be "dynamic content" which changes every time the page is requested. An example of this is a banner advert on a web page.

HTTP 404 Errors

When inspecting error details, you may see HTTP 404 errors reported (which relate to URLs in the User Journey that have no corresponding content on the web server).

If the web pages (thumbnails and full web pages) look correct, it is likely that the web site delivered a page containing URLs that caused the 404 errors but do not affect what the user sees. This is surprisingly common on very many web sites.

You can prove the URL is incorrect by pasting it into a web browser and confirming the browser also reports an error.

When you have done this, ideally you should report the problem to the web developers and ask them to correct the web page.

You should not remove the failing URLs from the User Journey – otherwise your performance test will not place the same load on the server as real world users do.

Internal System Errors (HTTP Error 5xx)

If these errors are reported, this means that the request supplied by StressTester caused an error to occur within the application.

Often, especially when the application you are testing is still under development, this error will reflect a genuine bug on the application server. However, the cause is also very often that the User Journey is not yet fully configured to send all hidden variables back to the server correctly.

If you have access to the application developers, the easiest thing to do is to ask them to look at the errors and identify the variables that are not being sent correctly.

Note:

Once you have identified the variable causing the problem, StressTester can be configured to handle it automatically using Auto-Correlated Dynamic Data.

to be continued in Next Issue

Back To Index

Biography

Juhi Verma is an Electronics Engineer working with Tata Consultancy Services (Mumbai) as a Tester. During her spell she has also worked as a programmer but she now prefers software Testing over programming as it's a dual fun, she says.

Testing is her passion and she enjoys participating and conducting testing related activities.

Juhi also works as a Team member at Tea-time with Testers. She can be contacted at her personal mail id juhi_verma1@yahoo.co.in or on Twitter @Juhi_Verma .

Do you think that even your own tool should be part of this unique section?*

Feel free to write us. Let the world know what your Tool can do !

To know more write to us at teatimewithtesters@gmail.com

* Conditions Apply

Biography

Sharmistha Priyadarshini is currently working as a Test Engineer at Tata Consultancy Services (Mumbai).

Sharmistha is die hard lover of Data Base testing as she finds is challenging. Being a programmer in past she now loves software testing too as it gives her more scope for analysis. Sharmistha can be reached via her mail id sharmistha105@gmail.com

Claim your Smart Tester of The Month Award. Send us an answer for the Puzzle and Crossword bellow b4 8th Sept 2011 & grab your Title.

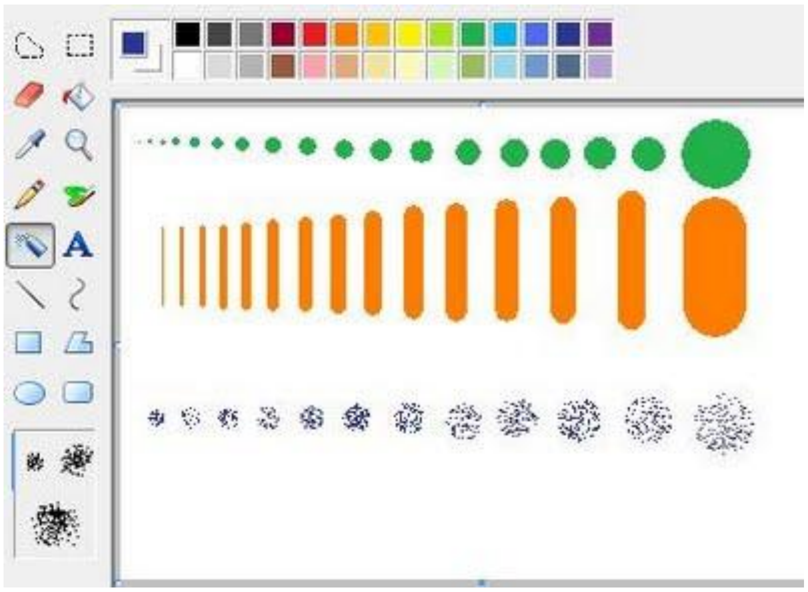
Send -> teatimewithtesters@gmail.com with Subject:Testing Puzzle

Gear up guys.......

It's Time To Tease your Testing Bone

Puzzle “Crash It”

"Create a sample file that makes Paintbrush (included in MS Windows) to crash when trying to open it"



Biography

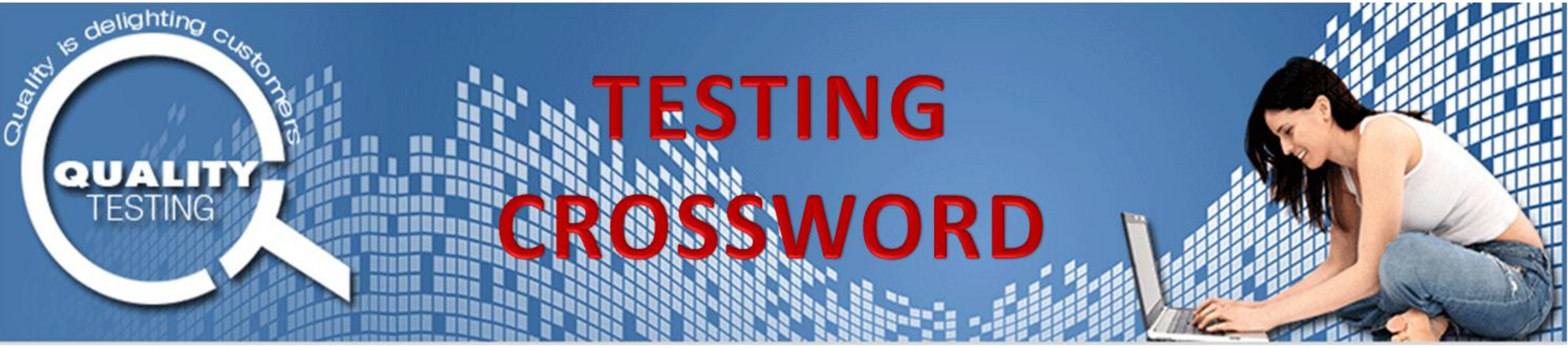
Blindu Eusebiu (a.k.a. Sebi) is a tester for more than 5 years. He is currently hosting European Weekend Testing.

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

He tweets as @testalways.

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

Back To Index



1			■	2	■	3		4	■
	■	■	5				■	6	
7	8				■		■		■
■		■	9					■	■
■		■	10		■		■	11	
■	12			■	■		■		■
■	13	■	14	■	15	■	■		■
16		■		■		■	17		
■		■		■		■	■		■
18					19				

Horizontal:

1. What is the short form of emergency bug fix (3)

3. Errors Successor (3)

5. It is a testing tool, which developed by Indian (4)

6. It is a predecessor of System Testing in short form (2)

7. It is a testing tool which uses Jquery (5)

9. QT Founder's first name (5)

10. Short form of You Are (2)

11. Testing against requirements in short form (2)

12. First three letters of most popular functional testing tool (3)

16. It is an object-oriented Unified Modelling Language software design tool intended for visual modeling in short form (2)

17. Ajax test runner for PHP tool in short form (3)

18. Alternate naming conversion for Issue (6)

19. It is a multi-Protocol load testing tool (5)

Vertical:

1. Last three letters of the popular Software Company (3)

2. A testing tool which uses Ruby language (5)

3. Processor uses this number system (6)

4. It integrates hardware, software, and data for capturing, managing, analyzing and displaying all forms of geographically referenced information in short form (3)

5. It is a visual technology to automate and test graphical user interface using image (6)

8. The world's largest marketplace for software testing services (5)

11. It standards for system testing in .Net (6)

13. Hierarchical representation of folder structure (4)

14. The world's largest professional association dedicated to advancing technological innovation and excellence for the benefit of humanity (4)

15. Lowest level of testing the application (4)

Answers for Last Month's Crossword:

R	E	G	R	E	S	S	I	O	N	S	
	R			W			N			T	
C	R	E	A	T	E		T	A	R	U	N
	O				X/T		E			B	
C	R	A	S	H	L	O	G	I	C	A	L
M			D				R				O
M			L		D		A		W	3	C
	S	E	C	U	R	I	T	Y	E		A
					E		I		T	S	L
	W	H	I	T	E	B	O	X		I	
	A				R	U	N			L	
	S	L	E	E	P	G	T	C	W	K	

Answers for Testing Puzzle:

1. "White Space" – by Srinivas Pattela
2. "~" - by Nishant Kumar Thakur
3. "2we9ez" – by Divya sp

Every Tester
who reads Tea-time with Testers,
Recommends it to friends and colleagues .
What About You ?
Image : vernhart

Our Testimonials

Hi Lalitkumar,

I'm impressed with Tea-time with Testers. The quality of your authors and the practical insights they share make **Tea-time with Testers** a great magazine and a welcome addition to the testing community.

Keep up the good work!

- **Justin Hunter**, CEO of Hexawise

www.Hexawise.com

Dear Justin,

Thanks for your kind words and inspiration.

-Editor

I had read Tea-time with Testers and have been impressed with both the quality of the writers and material that you have published.

Your passion for testing and fostering learning in the community is evident.

- Selena Delesie

Hi,
Here I wanna say that the portal seems interesting.
Yet to explore more...but am sure this will be a place to grab as much as we can..!
Glad to be here..!

– Heena

Looks very promising from the reviews and I am sure this will help my Testing skills to outstand and reach my goals.

- Shreeya Dhanpal

Great Great Great magazine! Great work guys and Thanks a lot !!

– Reena Joshi

Hi Editor...

I brows internet 24/7 through net in my mobile, my laptop, Office PC and always search something in Google to enhance my knowledge but NOW am sure that I don't need to search anywhere from now on...!

Tea-time with Testers Rocks! 👍👍👍

-Piramu S

Hi,

I read your magazine for this month it was worth to read it. It feels great after reading about Mobile QA Zone. Seems quite interesting & new concept.

- Priyanka

I love the magazine. Great stuff!

- Simon Reekie

I liked the idea of your magazine, and thanks for your great efforts and experience sharing.

- Mohamed Shalash

It is a great work / service to the testing community.

- Balaji Raaman

Heard about Tea-time with Testers, would like to share/learn and contribute

–Ajay Jain

Looks Very Interesting! - Christopher Davis

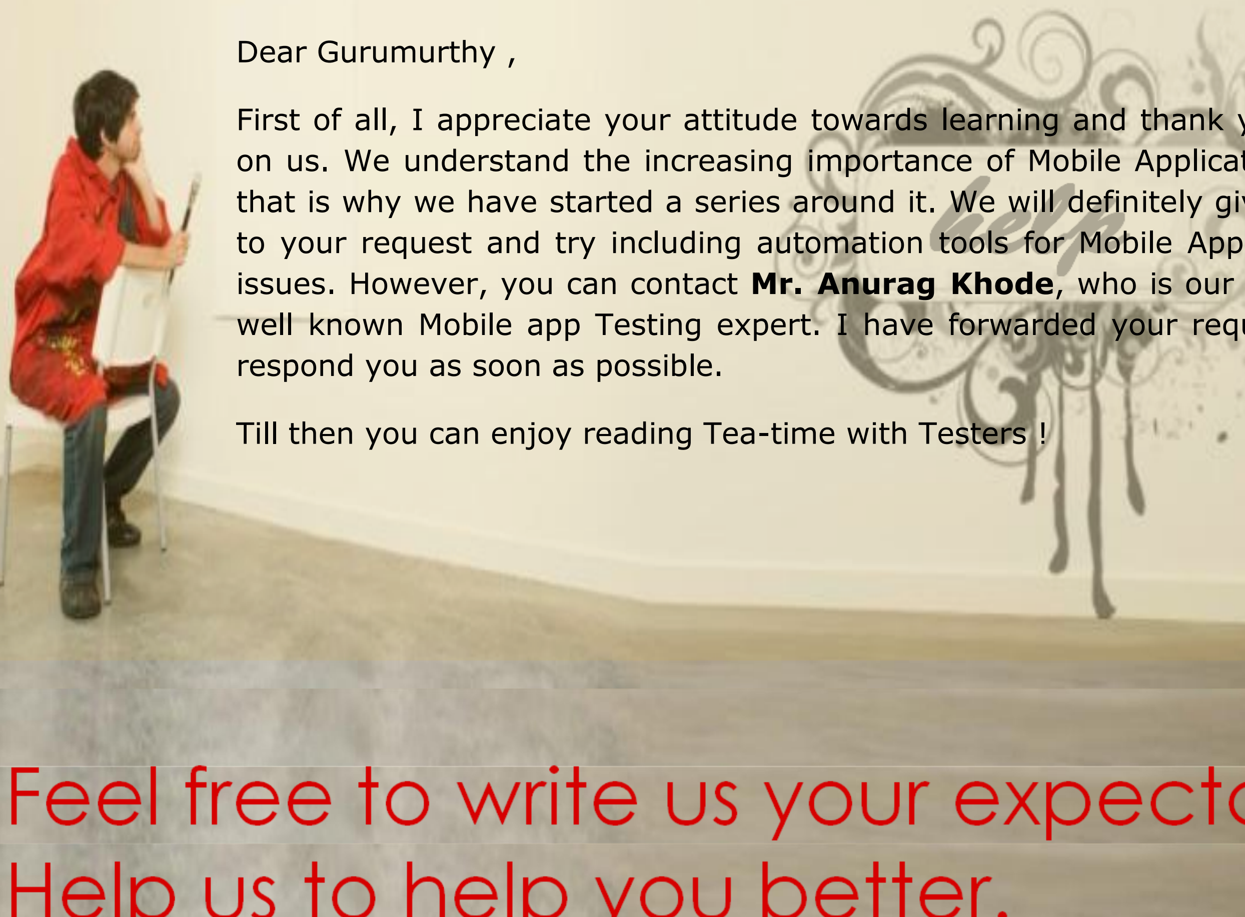
You ask...

We'll help...!

Hi ,

I would like to know Mobile Automation Testing tools and user guide to learn that. I have told my many friends to gain knowledge on automation of mobile project which will help us to lead better technical life. So I hope u could help here.

- Gurumurthy



Dear Gurumurthy ,

First of all, I appreciate your attitude towards learning and thank you for counting on us. We understand the increasing importance of Mobile Application Testing and that is why we have started a series around it. We will definitely give consideration to your request and try including automation tools for Mobile Apps in our coming issues. However, you can contact **Mr. Anurag Khode**, who is our Asst. Editor and well known Mobile app Testing expert. I have forwarded your request and he will respond you as soon as possible.

Till then you can enjoy reading Tea-time with Testers !

- Editor

Feel free to write us your expectations.
Help us to help you better.

We are just a mail away: teatimewithtesters@gmail.com

in nexxt issue

articles by -

Jerry Weinberg

T Ashok

Joel Montvelisky

Anurag Khode

Selena Delesie

David Burns

Ola Hylten

Olaf Lewitz

our family

Founder & Editor:

Lalitkumar Bhamare (Mumbai, India)

Pratikkumar Patel (Mumbai, India)

Lalitkumar

Pratikkumar

Editorial | Magazine Design | Logo Design | Web Design:
Lalitkumar Bhamare

Cover Page Image- Sagar Sankhe

Core Team:

Kavitha Deepak (Bristol, United Kingdom)
Debjani Roy (Didcot, United Kingdom)
Anurag Khode (Nagpur, India)

Anurag

Kavitha

Debjani

Mascot Design & Online Collaboration:

Juhi Verma (Mumbai, India)

Romil Gupta (Pune, India)

Juhi

Romil

Tech -Team:

Subhodip Biswas (Mumbai, India)
Chris Philip (Mumbai, India)
Gautam Das (Mumbai, India)

Subhodip

Chris

Gautam

|| Karmanye vadhikaraste ma phaleshu kadachna |
Karmaphalehtur bhurma te sangostvakarmani ||

To get **FREE** copy ,

Subscribe to our group at

Google™

Join our community on

facebook

Follow us on

www.teatimewithtesters.com