

# Tea-time with Testers

OCTOBER 2011 | YEAR 1 | ISSUE IX



**Jerry Weinberg**

Observing and Reasoning about Errors

**Anurag Khode**

Testing Checklist for Mobile Applications

**Michael Larsen**

Trading Money for Time

**Darren McMillan**

Lean Test Case Design

**T Ashok**

Seven Consecutive Errors= A Catastrophe

Investigating Bugs: A Testing Skills Study

A Long & Winding Road



**James Bach**

Software Testing Naturalist



**Matt Heusser**

Software Testing Pro





# TEA-TIME WITH TESTERS

**First Indian testing magazine to reach 76 countries in the world !**

Created and Published by:

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

Editorial and Advertising Enquiries:

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

This ezine is edited, designed and published by  
***Tea-time with Testers.***

No part of this magazine may be reproduced,  
transmitted, distributed or copied without prior written  
permission of original authors of respective articles.

Opinions expressed in this ezine do not necessarily  
reflect those of editors of ***Tea-time with Testers*** ezine.



# A Long and Winding Road

**Matt Heusser**

**O**ne fine evening, I got an e-mail from Tea-time with Testers Magazine.

I was flattered when Lalitkumar invited me to write Guest Editorial for Tea-time with Testers. He offered me an opportunity to write about my journey as a tester -- how I got to be here, and what's next.

When he made this offer, I realized that I had never told a story like that in print. Oh, I have written a lot about software testing. I have stories, and had mentioned some of them on my podcast, but the story of my career --that was just the beginning to be told.

## In the Beginning

In seventh grade, about age eleven, I had decided my profession. After reading a great deal of science fiction and fantasy, I had decided to be a writer.

By eighth, I was going to be a computer programmer.

In the beginning of ninth grade, I joined Civil Air Patrol, a sort of "Boy Scout" program sponsored directly by the United States Air Force. I liked the program because we wore military uniforms and did drill, but I never had that "wow airplane!" experience that many of my friends did. With little interest in becoming a pilot or an Air Force Pararescueman, I kept reading books, including books by Robert Heinlein and Douglas Hackworth. (My close friend in High School, Jake McGuire, loaned me both of those books that led to a direct and serious inquiry into the Combat Infantry.)

I was really interested in the infantry; I enlisted in the US Army Reserve at the Age of seventeen, signed up for Army ROTC, and, at some point, made it my goal in life to earn a commission in the United States Army before my twenty-first birthday.

Important Note: Fourteen Year old children rarely make serious, well-considered decisions.

The Army thing didn't work out; at some point I realized that, despite all the respect I have for the Armed Forces, I had no interest in training for combat with the hope of never having to use that training. (Fourteen year olds may dream of charging up a hill, shooting a machine gun, but within a few years I realized that the other guy would be shooting back. Or, as Douglas MacArthur put it "the soldier, above all other people, prays for peace, for he must suffer and bear the deepest wounds and scars of war.")

For that matter, I wanted to have a family, to actually, you know, *see them* and, if I might, to sleep in on Saturdays.

The good news is that even then, without knowing it, I was following Jerry Weinberg's old advice: "If you don't know what to do, make the decision that keeps your options open."

I did want to be a military officer, and that required a college degree, so I was in college, studying for a Bachelor's Degree in Mathematics with a Concentration in Computer Science.

In 1997, when I graduated, that was enough to get me a job as a computer programmer at Professional Computer Resources (PCR) in Grand Rapids, Michigan.

## Programming without a Safety Net

I spent a year and a half at PCR; it was a wonderful time. At the time, the company had perhaps thirty employees; the president, two managers, two administrators, and twenty-three programmers.

*...continued on [page 52](#) with Matt's interview*



# QuickLook



Crossword  
by



## Editorial

### What's making News?

### Tea & Testing with Jerry Weinberg

### Speaking Tester's Mind

Investigating Bugs : A Testing Skills Study -17

Trading Money for Time - 26

### In the School of Testing

Lean Test Case Design - 33

Testing Checklist for Mobile Applications - 40

Testing Intelligence: What do you pack when you go for Bug Hunt ? - 43

### T' Talks

Seven Consecutive Error = A Catastrophe - 47

### Tool Watch

### Testing Puzzle – S.T.O.M. Contest

### Interview with Matt Heusser

### Our Testimonials

### Family de Tea-time with Testers





September 2011

## Review: September issue of Tea-Time with Testers.

I've just finished reading September issue of Tea-Time for Testers, a free and useful e-magazine for the testers of the world. Here are a few comments I have to offer the magazine and its readers and potential readers.

- Selena Delesie's article is a gem. I wish all testers would copy the article and use it to lay out a learning plan for themselves. She leaves us no excuses of the sort "I don't have time for learning" or "there's simply no learning opportunities where I work." On flaw: she writes, "See my website for some books, websites and blogs I recommend." I tried the link, but couldn't find the list she mentions. I wish the link went directly to the list of recommendations that's hidden somewhere in her site.

- Anurag Khode's article on testing network connection speed is on a topic that interests me. It's highly specific and useful for that reason. I wish Anurag had given a few general conclusions that might help testers in other situations.

As it stands, the article itself is one model of setting up tests, but I suspect few testers will take advantage of that feature unless it's specifically pointed out.

- On the other hand, I look forward to Joel Montvelisky's articles precisely because they address the issue of applying intelligence. In this article, he tackles the application of intelligence in an area that many testers think doesn't require thinking at all: automated testing. He deserves a medal for courage, because I fear that some advocates of automated testing will lambast him for suggesting that you need to think once you've automated some tests.

Anyway, those are a sampling of the articles I enjoyed in the September issue. As usual, I read Tea-time with Testers from cover to cover (even though it has no covers). As for the issue as a whole, I always love the colorful illustrations throughout the issues.

- **Gerald M. Weinberg**

## Teach-Testing Campaign

This campaign is more than credible, I am a big fan of forward thinking Agile/Context driven ideas but also believe a grounding in more formal thought process such as ISEB/ISTQB is essential for testers and developers alike. Even for testers who will never use the old sequential models, that kind of grounding is imperative to know as even the newer, shorter, recently heralded models draw heavily from renowned best practice if you actually analyse the processes. I speak to too many young developers out of college who neither see the point in testing or testers. Our essential presence within the software development process is unknown to them (I'm in Canada), that has to change.

- **Martin Hore**

## Well researched & meaningful insight

I have been reading Tea-time with Testers for sometime now and I am happy that they are continuously striving to make this better day by day.

The articles are well researched and have a meaningful insight.

From the September 2011 issue, I liked Jerry's insight into errors which was so thoughtful yet so simple to understand. I am personally a big fan of Jerry's writings and hence might be biased in my opinion on the same :)

- Jerry / Matt / Selena / Ashok .... have been reading their works pretty often. But what really impressed me was reading from two new people whose writings I have not focused on before.

- Ola Hylten's writing on communication. I read something by Ola for the first time and I liked it. Communication is something so regular that we do in our day today life across every action and people connect we do but how many are able to perceive the science behind it.

- The other one was by Olaf Lewitz. So simple things and best represented for first time mentors.

- **Dr. Meeta Prakash**

## Connecting with great minds

Thank you for producing Tea-time with Testers and distributing it freely via the Internet. Your publication keeps me connected with some of the great minds of the software testing world and I appreciate it.

- **Ken Horovatin**



# What's making News?

- find out the latest happenings in the technology world

## Free software testing on USB for students to web developers

By Adrian Bridgwater on October 17, 2011

A bit of semi-random open source software searching is generally beneficial for the soul and spirit at least once a month. My most recent expedition in this vein led me to find Mantra, an open source browser-based security framework for penetration testing and security assessments.

Mantra then, or to use the original Tibetan script མན་ཏྲ་མེད་, is built/based on Mozilla's Firefox web browser (so it's cross platform) and can be taken "anywhere" on any rewritable media including memory cards, USB flash drives and portable hard disks.

According to its development team, "Mantra can be used for both offensive security and defensive security related tasks, which makes it incredible."

Mantra is part of the Open Web Application Security Project -- [OWASP](#).



OWASP itself describes Mantra as follows, "Mantra is a collection of free and open source tools integrated into a web browser, which can become handy for students, penetration testers, web



application developers, security professionals etc. It is portable, ready-to-run, and compact and follows the true spirit of free and open source software."

**ANALYSIS --** As the Mantra team now strives to create an ecosystem for hackers based on browser technology, they will need to be clear on their messages as to the reason and need for testing and why bugs originate in the first place.

This may be the reason for calling the project Mantra and trying to bring a little perception of human behaviour to the deeper themes associated with software testing i.e. as software systems now enter ever more deeply complex constructs, humans have only limited ability to manage complexity -- this means that, in complex systems at least, software bugs, defects and errors can never be eradicated completely.

Until we reach coding Nirvana निर्वाण that is of course...

## IT cos outsource freshers' training

DIKSHA DUTTA & KIRTIKA SUNEJA

On Saturday, Oct 15, 2011 for **The Financial Express**

---

**New Delhi:** To resolve the problem of employability and reduce the ever-growing training costs of freshers, industry majors like IBM and Cognizant have started training students right from their days at engineering college.

This is to ensure an employable fresher with special skills when he is ready to join the \$70 billion IT industry. Though the training may or may not get them jobs in top companies, it definitely increases chances of getting a job in the tech sector and also increases the placement rate of tier-II and tier-III institutes like Punjab Technical University, Apeejay College of Engineering and Noida Institute of Engineering.

The training starts from the fourth semester itself and the students are certified in the fourth semester. These skills are as niche as software testing, which are not considered as very high-end jobs in the industry as they make engineers eligible for a salary ranging from R12,000-20,000. By the time these students graduate from college, they are ready to be on the job.

Sample this: Cognizant, the country's third largest software exporter hired 600 students trained on software testing from tier-II and tier-III institutes, claims QAI, a consulting and workforce development organisation which trains students on such skills as and when demanded by different tech majors. QAI has trained 4,000 engineers overall on software testing till now. At present, the country produces only 3,000 testers, but the nation will soon need 25,000-30,000 testers each year.

"Companies are in constant need for skilled engineers and they are also ready to give conditional offer letters to students, depending on their need. We train the students as per the demand of tech majors," said Mohandas Menon, vice-president, Education QAI.

Once trained, these students have a better chance to clear the interviews of companies like IBM and Cognizant.

"More than 90% of the students trained by us get a job within the industry. This automatically increases the conversion rate of campus placements in tier II and tier III institutes," added Menon.

Himanshu Goyal, country manager, IBM Career Education corroborated, "Today, the industry faces a challenge in training employees on righteous skills like software testing. We are helping students getting trained on such skills in collaboration with QAI. And if we have a requirement for such talent, IBM would definitely consider these trained students for a job with IBM".

The co-designed programmes by QAI and IBM Career Education will be offered on campus across India as a blended learning programme supported by world class courseware and internationally certified faculty members.

Research firm Everest attributes this trend to the recession which led to the pruning of delivery layers deployed by most service providers. Firms shifted as much as 20% of their offshore headcount to tier-II and tier-III cities which helped them provide lower pricing without sacrificing margins.

For more updates on Software Testing, visit → [Quality Testing - Latest Software Testing News!](#)

Announcing...

## Smart Tester of the Month Awards !!!

### ❖ Winners for Testing Crossword :

Sonal Agarwal , Noida .

Devaganaraja Gopalasetty, Hyderabad

### ❖ Winners for Testing Puzzle :

Prakhar Jain ( Special Mention )

V. Vinod Kumar

Ashwini Sadashivam

Shanalene Silva

Nixon Josephraj

Lakshmikanth Mohan

Rangarajan Agasthian

Dharshini Sivalingam

Harmya Khandhadia

# Congratulations !



# Look Who's Rising



Nine Months

7000+ Readers

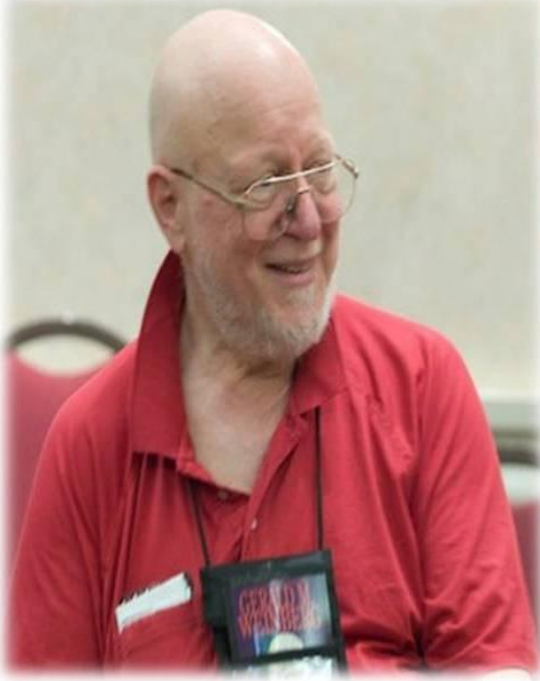
76 Countries

ONE Magazine

## TEA-TIME WITH TESTERS

Subscribe here Right Away to get our all Issues for FREE

# Tea & Testing



with

Jerry Weinberg

## Observing and Reasoning About Errors (Part 3)

### Why is someone working late?



A programming team manager told me, "Josh is my best programmer. The reason he starts work in the afternoon and stays late at night is so he won't be disturbed by the less experienced programmers."

It turned out that Josh was so ashamed of the poor quality of his work that he didn't want anyone to see how much trouble he was having.

### Who knows what's right and wrong?

Another team leader told me, "Cynthia is angry because I showed her what was wrong with her program, and how it should have been done in the first place. I suppose you're going to tell me I have to learn to be



more tactful." Cynthia showed me the program and what the team leader had said was wrong. It wasn't wrong at all. Cynthia said, "What ticks me off is working under a boss who's not only technically illiterate, but doesn't know how to listen. He approaches every problem with an open mouth."

### Which process is eliminating problems?

A project manager told me, "We've abandoned technical reviews in this project. They were valuable at first, and we found a lot of problems. Now, however, they don't find much trouble— not enough to justify the expense." As it turned out, the reason there were no problems was that the programmers were conducting secret reviews, to hide their errors from the manager, who berated anyone whose product showed errors in the review. They had not abandoned technical reviews; they had abandoned the practice of telling their manager about their technical reviews.

Feedback controllers use observations of behavior to decide upon actions to eliminate undesired behaviors. They feed these actions back into the system and thus create a negative feedback loop to stabilize the system, as shown in Figure 1-1.

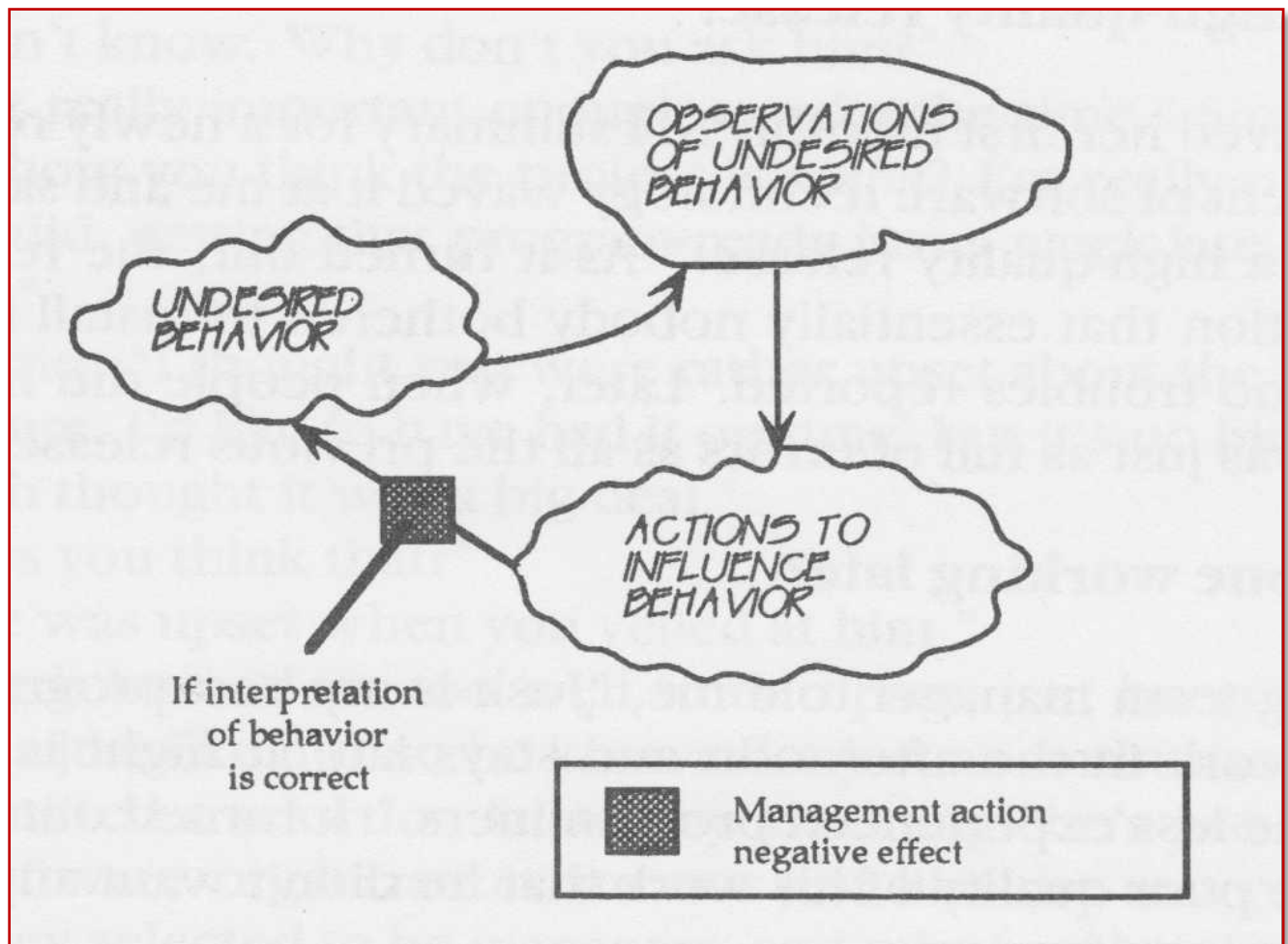


Figure 1-1. The feedback controller uses observations to decide upon actions to stabilize the system's behavior.

When the feedback controllers gets the meaning of the observation backwards, however, the designed actions create a positive feedback loop, actually encouraging the undesired behavior, as shown in Figure 1-2.

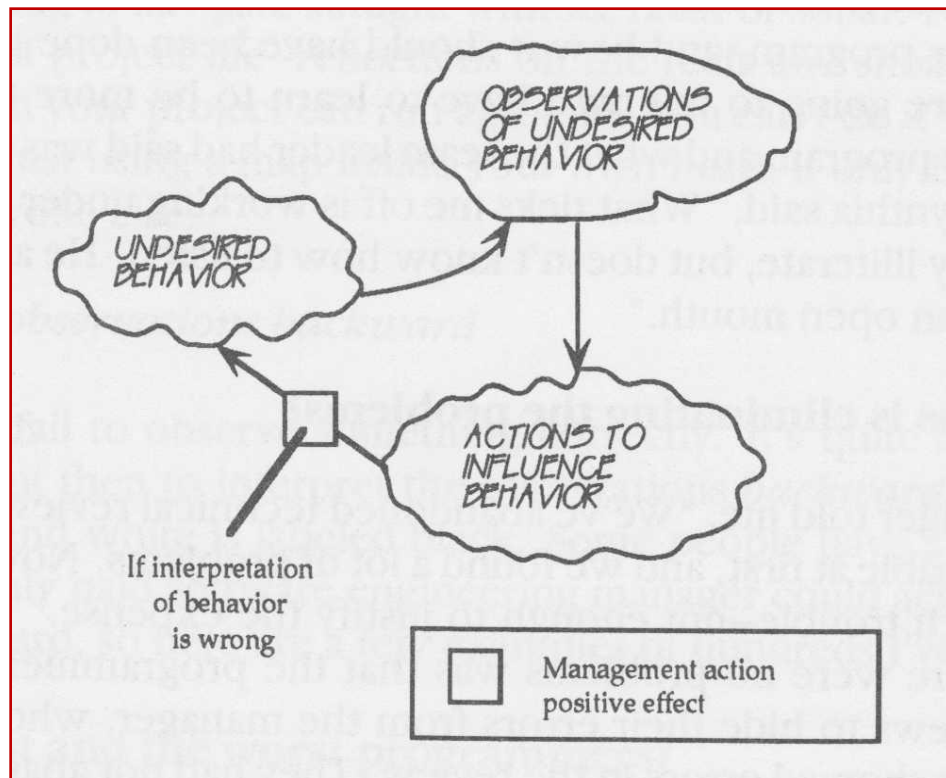


Figure 1-2. Getting the meaning of an observation backward creates an intervention loop that promotes what it should discourage and vice versa.

### 1.3.3. The Controller Fallacy

The example of eliminating technical reviews illustrates another common observational fallacy. Even if it were true that reviews were no longer finding errors, would that be the reason to abandon them? Technical reviews serve many functions in a software project, but one of their principal functions is to provide feedback information to be used in controlling the project. In other words, they are part of the controller's system.

It is the nature of feedback controllers to have an inverse relationship to the systems they attempt to regulate.

- We spend money on a thermostat so we won't spend extra money on fuel for heating and cooling.
- We keep the fire department active so that fires will be inactive.
- We put constraints on the powers of government so that government won't put unnecessary constraints on the governed.

One result of this inverse relationship is this

The controller of a well-regulated system may not seem to be working hard.



But managers who don't understand this relationship often see lack of obvious controller activity as a sign that something is wrong with the control process. This is the Controller Fallacy, which comes in two forms:

If the controller isn't busy, it's not doing a good job.

If the controller is very busy, it must be a good controller.

Managers who believe the second form are the ones who "prove" how important they are by being too busy to see their workers.

The first form applies to the reversed technical review observation. If the technical reviews are not detecting a lot of mistakes, it could mean that the review system is broken. On the other hand, it could also mean that the review system is working very well, and preventing faults by such actions as

- motivating people to work with more precision
- raising awareness of the importance of quality work
- teaching people how to find faults before coming to reviews
- detecting indicators of poor work, before that work actually produces faults
- teaching people to prevent faults by using good techniques they see in reviews

#### **1.4. Helpful Hints and Suggestions**

- Usually, there are more failures than faults, but sometimes, there are faults that produce no failures — at least given the usage of the software up until the present time. And sometimes it takes more than one fault to equal one failure. For instance, there may be two that are "half-faults," neither of which would cause trouble except when used in conjunction with the other. In other cases, such as in performance errors, it may take an accumulation of small faults to equal a single failure. This makes it important to distinguish between functional failures and performance failures.

- Proliferation of acronyms is a sign of an organization's movement towards Pattern 2, where name magic is so important that a new name confers power on its creator. Care in designing acronyms is a sign of an organization's movement towards Pattern 3, where communication is so important

- You can almost count on the fact that first customers aren't like later ones. Managers often commit a selection fallacy in planning their future as software vendors based on initial favorable customer reactions to a software system. The first customers are first because they are the ones the requirements fit for. Thus, they are very likely to be "like" the original designer/developers. Developers and customers communicate well, and think alike. This is not so as the number of customers grows, and explicit processes must be developed to replace this lost "natural" rapport.

- Selection fallacies are everywhere. You can protect yourself by wearing garlic flowers around your neck, or else by asking, whenever someone presents you with statistics to prove something, "Which cases are in your sample? Which cases are left out? What was the process by which you chose the cases you chose?"

## 1.5 Summary

1. One of the reasons organizations have trouble dealing with software errors is the many conceptual errors they make concerning errors.
2. Some people make errors into a moral issue, losing track of the business justification for the way in which they are handled.
3. Quality is not the same thing as absence of errors, but the presence of many errors can destroy any other measures of quality in a product.
4. Organizations that don't handle error very well also don't talk very clearly about error. For instance, they frequently fail to distinguish faults from failures, or use faults to blame people in the organization.
5. Well functioning organizations can be recognized by the organized way they use faults and failures as information to control their process. The System Trouble Incident (STI) and the System Fault Analysis (SFA) are the fundamental sources of information about failures and faults.
6. Error-handling processes come in at least five varieties: detection, location, resolution, prevention, and distribution.
7. In addition to conceptual errors, there are a number of common observational errors people make about errors, including Selection Fallacies, getting observations backwards, and the Controller Fallacy

## 1.6. Practice

1. Here are some words I've heard used as synonyms for "fault" in software: lapse, slip, aberration, variation, minor variation, mistake, oversight, miscalculation, blooper, blunder, boner, miscue, fumble, botch, misconception, bug, error, failure. Add any words you've heard to the list, then put the list in order according to how much responsibility they imply on the human beings who created the fault.
2. When an organization begins the systematic practice of matching every failure with a known fault, it discovers that some failures have no corresponding fault. In Pattern 3 organizations, these failures are attributed to "process faults"—something wrong with their software process that either generated fictitious failures or prevents the isolation of real ones. List some examples of process faults commonly experienced in your own organization, such as careless filling out of STI records.
3. For a week, gather data about your organization in the following way: as you meet people in the normal course of events, ask them what they're doing. If it has anything to do with errors of any kind, make a note of how they label their activity—debugging, failure location, talking to a customer, or whatever. At the end of the week, summarize your findings in a report on the process categories used for error work in your organization's culture.
4. Describe a selection fallacy that you've experienced. Describe its consequences. How could a more appropriate selection have been made?



# 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](mailto:hardpretzel@earthlink.net) or on twitter @JerryWeinberg

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

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](#).



**TTWT Rating:** ★★★★★

A photograph of a green conical pendulum bob hanging from a thin wire. The bob is positioned over a surface of light-colored sand. In the sand, there is a large, circular, swirling pattern that resembles a fingerprint or a stylized mandala. The text "Speaking Tester's Mind" is overlaid on the image in a large, blue, serif font with a white outline.

# Speaking Tester's Mind

- straight from the author's desk



# Investigating Bugs: A Testing Skills Study

C  
O  
V  
E  
R  
  
S  
T  
O  
R  
Y



*By James Bach*

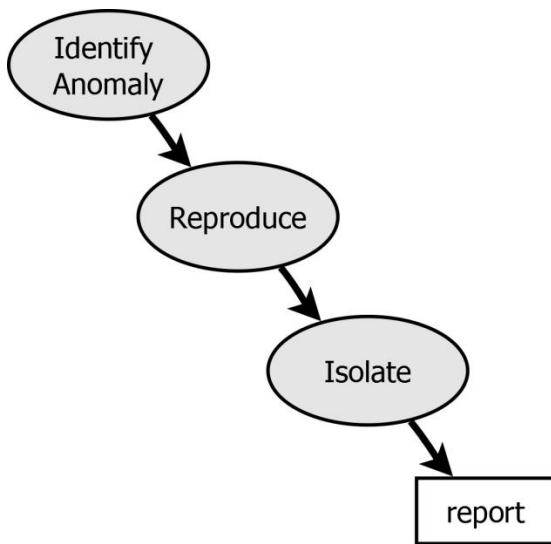
Ask any experienced tester how he does his work, and the answer is likely to be extremely vague ("Um, you know. I use my experience to... Um... black box the test case plan and such..."), or extremely false ("Our testing consists of detailed formal test procedures that are derived from written requirements"). Forget about bad testers, even *good* testers are notoriously bad at explaining what they do. Doing testing, describing testing, and teaching testing are all different things. No wonder that the IEEE testing standards are a joke (a very old joke, at this point), and based on talking with people involved in the upcoming ISO standard, it will be no improvement.

If we truly wish to develop our craft toward greater professional competence and integrity, then before we can worry how testing should be, we must be able to say how it is. We must study testers at work. Let me illustrate.

Years ago, I was hired by a company that makes printers to help them develop a professional testing culture. Instead of bringing in all my favorite testing practices, I started by observing the behavior of the most respected testers in their organization. I divided my study plan into segments, the first of which was bug investigation.

The organization identified one team of three testers (two testers and a test lead) that had a great reputation for bug investigation. This team was responsible for testing paper handling features of the printer. I wanted to see what made them so good. To get the most accurate picture of their practices, I became a participant-observer in that team for one week and worked with them on their bug reports.

## Stated Procedure for Investigating Bugs



I sat down with George, the test lead, and asked him how he did bug investigation. He said, "Don't ask me to change anything."

"Good news, George," I replied. "I'm not that kind of consultant." But after some feather smoothing, he did answer:

1) *Notice if test automation finds a problem.* The automation system alerts the testers that something did not perform as expected.

2) *Reproduce the problem.* The tester executes the test again to recreate the symptoms of the problem.

3) *Isolate the problem.* The tester edits the test script, cutting it down to the minimum required to exhibit the problem.

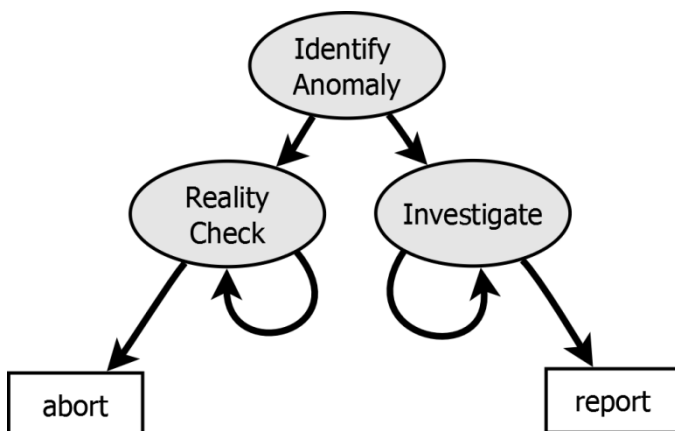
4) *Pass the problem to the test lead for investigation and reporting.* The tester delivers the edited test script to the test lead. The test lead investigates the problem to determine, as best he can, its dynamics and causes. The test lead then writes the bug report and submits it.

This description is typical of how testers claim to investigate and report bugs. It's only unusual in that the test lead performs the investigation and writes the actual bug report. However, there was another way in which this description was unfortunately typical: *it's not true.*

I knew it couldn't be true, because it's a process description that anyone in that company would claim to use, yet only this team was respected for the quality of its work. There must be something more to their process that I wasn't being told.

Sure enough, during my observations and further conversations with the team, I found the actual process used in the team to be much more sophisticated and collaborative than their stated process. Their actual process ranks among the best I have seen at any company. (I first said that in the year 2000, and it remains true in the year 2011).

## Observed Process of Investigating Bugs



What I observed in practice was an exploratory investigation carried out by the whole team. When an anomaly worth investigating was spotted by a particular tester, the other two came over and they engaged together in two parallel loops of inquiry: *investigating the bug* and *questioning the status and merit of the investigation itself*. There were two possible outcomes, aborting the investigation or reporting the bug. The actual bug report was written by the test lead.



## Part I: Identification

### 1. Notice a problem (during an automated test or any other situation).

It is a general practice in the industry to construct tests that have specific expected results. This team took that idea further. Although they did establish certain specific expectations in advance, they also looked at whatever happened, as a whole, and asked themselves if it made sense. This extended even to the pattern of clicks and whirrs the printer made as it processed paper, and the timing of messages on the control panel. I call this the *Explainability Heuristic*: any unexplained behavior may be a potential bug, therefore we attempt to explain whatever happens.

### 2. Recall what you were doing just prior to the occurrence of the problem.

### 3. Examine symptoms of the problem without disturbing system state.

Prior to starting a full investigation of a problem that may be difficult to reproduce, the testers capture as much volatile information as they can about it. This includes reviewing their actions that may have triggered the problem and examining the problem symptoms while disturbing the state of the printer as little as possible.

During identification, the tester's transition from a defocused behavior of observing whatever might be important to the focused task of investigating specific states and behaviors.

## Part IIa: Reality Check Loop

The tester must decide whether to pursue the investigation or move back into open testing. So, prior to launching a bug investigation, and repeatedly during the investigation, these questions are asked.

1. Could this be tester error?
2. Could this be correct behavior?
3. Is this problem, or some variant of it, already known?
4. Is there someone else who can help us?
5. Is this worth investigating right now?
6. Do we know enough right now to report it?



The test lead is usually consulted on these questions before the investigation begins. But testers apply their own judgment if the test lead is not available. Contrary to George's first description of how his team worked, the testers on his team used initiative and routinely made their own decisions about what to do next.

In the event that an investigation is suspended because of the difficulty of reproducing it, it may still be reported as an intermittent problem. Whether or not it's reported, the testers will preserve their notes and be on the lookout for the problem as they continue testing. Some investigations go on like that for months.



## Part IIb: Investigation Loop

Once it's determined that the anomaly is worth looking into, the investigation begins in earnest. As I observed them, bug investigations were not a linear execution of predefined steps. Instead, they proceeded as an exploratory process of gathering data, explaining data, and confirming explanations. The exploration was focused on reproducing the problem and answering certain key questions about it. When they were answered well enough, or when the amount of time and energy spent on the problem exceeded its importance (that's what the reality check loop is all about) the investigation ended and the bug was reported. Sometimes investigations continued after making an initial report. This was done so that the developers could begin to work on the bug in parallel with testers' efforts to give them better information.

The investigation process is marked by a series of focusing questions that are repeatedly asked and progressively answered:

### 1. How can the problem be reproduced?

The testers not only reproduce the problem, they try to find if there are other ways to make it happen. They progressively isolate the problem by discerning and eliminating steps that are not required to trigger it. They look for the most straightforward and general method of making it happen. They also seek to eliminate special conditions or tools that are not generally known or available, so that anyone who reads the bug report, at any later time, will have the ability to reproduce the problem.

### 2. What are the symptoms of the problem?

Apart from identifying and clarifying its obvious symptoms, the testers are alert for symptoms that may not be immediately obvious. They also look for other problems that may be triggered or exacerbated by this problem.

### 3. How severe could the problem be?

The testers try to analyze the severity of the problem in terms of how it would affect a user in the field or create a support issue for the company. They look for instances of the problem that may be more severe than the one originally discovered. They look for ways to reproduce the problem that are most plausible to occur in the field.

The testers also consider what this kind of problem may indicate about other the potential problems not yet discovered. This helps them assure that their test process is oriented toward areas of greatest technical risk.

### 4. What might be causing the problem?

The most interesting element I observed in the team's process of bug investigation is their application of technical insight about printer mechanisms (both hardware and firmware) to guide their investigation of the problems. In the course of investigation, the testers did not merely manipulate variables and factors arbitrarily. They investigated systematically based on their understanding the most likely variables involved. They also consulted with developers to refine their understanding of printer firmware dynamics.

Although there is no set formula for investigating problems, I observed that the testers relied upon their knowledge of printer mechanisms and their experience of past problems to organize their





investigation strategy. So, maybe that's the formula: learn about how the printer works and pay attention to patterns of failure over time.

### Part III: Reporting

Although I participated in bug investigation, I did not personally observe the process of writing a bug report in this team. The testers reported that they sometimes wrote a draft of the bug report themselves, but that all reports were edited and completed by the test lead.

### Supporting Factors that Make the Process Work

#### *Bug Investigation Philosophy*

Apart from the process they follow, I found that there was a tacit philosophy of bug investigation in the team that seems to permeate and support their work. Here are some of the principles of that philosophy:

- We expect testers to learn the purposes and operational details of each test.
- We expect testers, over time, to gain a comprehensive expectation of the behavior of the product, and to follow-up on any anomalous behavior they detect at any time.
- Each bug is investigated by all members of the team.
- Bug investigation is primarily our job, not the developers. If we do our job well, then developers will be able to do their jobs better, and they will respect us for helping them.
- Testers should develop and use resources and tools that help in bug investigation.
- Ask for help. Someone else may know the answer or have an important clue. Seek advice from outside the team.
- We expect testers to use initiative in investigation and consult with the test lead as they go.

#### *Individual Initiative and Team Collaboration*

During the period I observed, the testers in the team treated each bug investigation as a group process. I had seen this before, and rarely since. They also consulted with testers outside their team, and with developers. Their attitude seemed to be that someone in the next cube may have information that will save them a lot of time and trouble.

The testers also showed personal initiative. They did not seem worried about crossing some forbidden line or running afoul of some corporate rule during the course their investigation. They appeared to take ownership of the problems they were investigating. The test lead told me that he encouraged initiative in his testers, and that he expected the testers, over time, to learn how all the tests worked and how the printers worked. In separate interviews, the testers confirmed that sentiment, and stated that they felt that the resulting working conditions in their team were better than in most other teams they had served on at that company.



## Observed Skills

I saw each of the following skills exhibited to some extent in each of the testers in the team. And in my opinion, the method of the investigation used in the team requires competence in these skills.

- *Skepticism.* Skepticism might be called the fear of certainty. It can be seen as central to the challenge of thinking scientifically; thinking like a tester. Good testers avoid sweeping claims about the product, because any claim may be refuted with the execution of the next test.
- *Performing an open investigation.* An open investigation is a self-managed investigation with general goals and few explicit constraints. An open investigation involves coordinating with clients, consulting with colleagues, collecting information, making conjectures, refuting or confirming conjectures, identifying issues, discerning and performing tasks, and reporting results. An open investigation in conjunction with testing is commonly called “exploratory testing.”
- *Understanding external and internal product functionality.* Bug investigation requires a sufficient understanding of both external and internal workings of the technology. This knowledge is gained over time and over the course of many investigations, and through studying documentation, exploratory testing, or by observing other testers at work.
- *Consulting with developers or other testers.* Vital information needed to investigate problems is scattered among many minds. Good testers develop an understanding of the network of people who may be able to offer help, and know to approach them and efficiently elicit the information they need. In the case of developers, testers need the ability to discuss and question software architecture.
- *Test factoring.* When anomalous behavior is observed in the product, the ability to isolate the factors that may be causing that behavior is at the heart of the investigation process. This includes insight about what factors may be causally related, the ability to form hypotheses about them and to relate those hypotheses to observable behavior.
- *Experiment design.* Testers must be able to reason about factors and find methods of controlling, isolating, and observing those factors so as to corroborate or refute hypotheses about the product.
- *Noticing problems.* A tester can know how the product should function and yet still not notice a malfunction. Being alert for problems, even in the middle of investigating other problems, and even in the absence of an explicit and complete specification, is a skill by itself. This requires a good knowledge of applicable oracles, including tool-based oracles.
- *Assessing problem severity.* This requires understanding the relationship between the technology, the hypothetical user, the project situation, other known problems, and the risk associated with problems that may lie hidden behind the one being investigated. This skill also requires the ability to imagine and articulate problem severity in terms of plausible scenarios.
- *Identifying and using technical documentation.* Bug investigation often requires spot learning about the product. With printing technology, that can mean poring through any of thousands of pages of technical documents. Testers need to know where and how to find relevant information.
- *Recording and maintaining information about problems.* The testers must deliver information about a problem in an organized and coherent form in order for the test lead to confirm it and write the report. This includes the ability to make and maintain notes.
- *Identifying and using tools.* Tools that may aid testing are scattered all about. Enterprising testers should be constantly on the lookout for tools that might aid in the execution of tests or diagnosis of problems. Testers must have the ability and initiative to teach themselves how to use such tools.

- *Identifying similar known problems.* In order to know if a similar problem is already known, the testers must know who to check with and how to search the bug tracking system. This also requires enough technical insight to determine when two apparently dissimilar symptoms are in fact related.
- *Managing simultaneous investigations.* Rarely do we have the luxury of working on one thing at a time. That goes double when it comes to investigating intermittent problems. Such investigations can go on for weeks, so testers must have the ability to maintain their notes and report status over the long term. They must be able to switch among investigations and not let them be forgotten.
- *Escalation.* Since these investigations are largely self-managed, it's important to know when and how to alert management to issues and decisions that rightly belong at a higher level of responsibility.

## Case Study: The Frozen Control Panel

This is an example of an actual investigation in the team that took place while I watched. It appears to be typical of other investigations I had been told about or personally observed. The important aspect of this case is not the conclusion— we could not reproduce this problem— but rather the initiative, teamwork, and resourcefulness of the testers. This investigation is documented in as much detail as we could remember in order to provide a feel for richness and flow of an exploratory testing process.

1. While Clay was running one of the paper handling tests, he encountered a printer lockup. Clay called Ken and James over to observe and assist.
2. Clay had been running an automated test script that included many steps. After executing it once he started it again. This time, it began executing, then stopped, apparently waiting for a response from the printer. At that point Clay noticed that the printer was frozen.
3. Clay asked Ken if he knew about the problem and whether he thought the problem was worth investigating.
4. Without resetting the printer, Ken examined the surrounding symptoms of the problem:
  - Check control panel display (display showed "Tray 5 Empty" continuously).
  - Check ready and data light status (both were lit and steady).
  - Open and close a tray (display did not react; engine lifted the tray).
  - Open and close a door (display did not react; engine performed paper path check).
  - Try control panel buttons (display did not react to any buttons).
5. Ken and Clay examined the test output in the terminal window and discovered that the test harness tool had stopped during its initialization, before any script code had been executed.
6. After a brief conference, Ken and Clay decided that the problem was worth investigating and conjectured that it may be due to an interaction between the timing of control panel display messages and messages sent to the printer.
7. Ken performed a cold reset of the printer.
8. Clay restarted the test tool. The problem did not recur.
9. Clay edited the test script down to the last few operations. He executed the modified script several times. The problem did not recur.



10. To test the hypothesis that the problem was related to the timing of alternating "READY" and "TRAY 5 EMPTY" displays on the control panel, Ken and Clay coordinated with each other to start executing the test tool at various different timings with respect to the state of the control panel display. No problem occurred.
11. We went to see a firmware developer on the control panel team, and asked him what might account for these symptoms. He seemed eager to help. He suggested that the problem might be a deadlock condition with the engine, or it might be a hang of the control panel code itself. He also suggested that we review recent changes to the firmware codebase, and that we attempt to reproduce the problem without using the test tool. During the course of this conversation, the developer drew some basic architectural diagrams to help explain what could be going on. We questioned him about the dynamics of his diagram.
12. The developer also conjectured that the problem could have been leftover data from a previous test.
13. Then we went to see a fellow who once supported the test tool. With his help, we scrolled through the source code enough to determine that all the messages displayed by the tool before it halted were issued prior to contacting the printer. Thus, it was possible that whatever happened could have been triggered by the first communication with the printer during tool initialization. However, we were unable to locate the tool routine that actually communicated with the printer.
14. Because the control panel locked up with the data light on, we knew that it was unlikely to have been in that state at the end of the previous successful test case, since that case had reported success, and left no data in the printer.
15. We looked for a way to eavesdrop on the exact communication between test tool and the printer, but found there was no easy way to do that.
16. We called upon another tester, Steve, for help, and together we wrote a shell script, then a Perl script, that endlessly looped while executing the test tool with an empty script file. At first we thought of introducing a random delay, but Clay argued that a fixed delay might better cover the timing relationships with the printer, due to the slight difference between the fixed time of the test and the presumably fixed response time of the printer.
17. We ran the script for about an hour. The printer never locked up.
18. While watching the control panel react to our script, Clay saw a brief flicker of an unexpected message on the display. We spent some time looking for a recurrence of that event, but did not see one.
19. We then went to visit a control panel tester to get his ideas on whether a problem like ours had been seen before, and how important a problem it could be. He advised us that such a problem would be quite important, but that he knew of no such problem currently outstanding.
20. Clay independently searched the bug tracking system for control panel problems, and found nothing similar, either.
21. After several hours of all this, we were out of easy ideas. So, we called off our investigation until the test lead returned to advise us.



## The Study of Skill is Difficult

It's quite difficult to study the anatomy of a practice, and the skills that practice requires. You can't know at first exactly what to watch, and what to ignore. Anthropologists learn to watch behavior for long periods of time, and to relentlessly consider the possibility of researcher bias. And just the act of studying a set of skills makes people nervous and possibly changes their behavior. I had to agree not to release any information about the progress of my study until I cleared it with the people I was studying.

Still, even a modest one-week study like this one can have profound positive effects on the team. When I gave this report to the team for approval, the team was a bit stunned at how much my description differed from their self-description. One of the testers asked me if he could staple it to his résumé. Perhaps there is even more depth to the skills of bug investigation than I have identified so far, but this is the sort of thing we must begin to do in our field. Observe testers at work and go beneath the grossly general descriptions. See what testers really do. Then maybe we can truly begin to build a deep and nuanced vision of professional software testing.



**James Marcus Bach** is a software tester, author, trainer and consultant. He is a proponent of Exploratory testing and the Context-Driven School of software testing, and is credited with developing Session-based testing.

His book "**Lessons Learned in Software Testing**" has been cited over 130 times according to Google Scholar, and several of his articles have been cited dozens of times including his work on heuristics for testing and on the Capability Maturity Model. He wrote numerous articles for IEEE Computer.

Since 1999, he works as independent consultant out of Eastsound, Washington.

He is an advisor to the Lifeboat Foundation as a computing expert.

Follow James on Twitter @jamesmarcusbach or know more about his work on [satisfice.com](http://satisfice.com)

# Trading Money for Time :

## When Saving Money Doesn't (And When It Does) - Part 1



***By Michael Larsen***

*How you spend your time is more important than how you spend your money.  
Money mistakes can be corrected, but time is gone forever.*

-David Norris.

In any endeavor I participate in, I have choices. In many of life's transactions, I can spend money to have something done, or I can spend the time to do that same thing. For some things, it's worth it to invest the time to make up or offset the amount of money to be spent. At other times, there is a level of involvement and a required expertise that makes not spending the money a barrier to achieving my goals. There is a tradeoff; money for time, or time for money. The problems arise when we don't give both of each of these areas proper consideration. In the world of software development, containing costs is important, but so is delivering a good quality product at the right time. Containing costs when a project is on time is a good thing, but losing time and not delivering a product when scheduled can cost both time and money, often in the lost revenue from dissatisfied customers. At the furthest extreme, the resulting loss of time could result in legal action for not being able to deliver on commitments.

This article examines the tradeoffs between money and time, and demonstrates examples where time investment has meant financial gain, and where lack of understanding of time has negated or even negatively affected a company's finances.



## A New Project Gets Underway

It's time for another software project to begin. Each time I go through this process, I have a similar set of experiences.

Each time I need to:

- Address physical infrastructure needs
- Determine where to set up machines
- Gather and install software applications to support the testing needs
- Consider the scope and types of testing that will be required

Physical infrastructure needs include buying and setting up server and client system and networking equipment. It may also involve internal or external hosting considerations, plus the adequate securing of these resources. When we consider physical machines, potential lab space for the devices and the need for adequate cooling must be considered. Our software application under development also has infrastructure requirements. SDK's and IDE's are needed for software development. We must configure and maintain file services, database services, web services, etc. We also have to consider deployment of the application(s) for testing and for general use. The specific testing areas (and the software testing tools that we utilize) must also be considered. Unit testing. Functional testing. Load testing. Performance testing. Para-functional and human factors/user experience testing. The number of areas to manage and maintain is dizzying!

I have witnessed various companies and their management teams wrestle with the goal to improve the testing and development process. These goals may be to speed up delivery, or add testing coverage for initiatives that haven't adequately been tested. Many management teams also provided fanfare and cheering about the time savings due to some new enhancement. It might be a new tool to automate all of the regression testing. It could be a new server to run a battery of virtual environments. A robust versioning and back-up system to keep all versions of files updated, tagged, secure and available would streamline the application lifecycle.

These are admirable goals, and when implemented, the net result can be more productive testing. Work can be completed more quickly.

All of these initiatives cost us something. To set up any of these enhancements, our teams have to spend something to bring them to fruition. With organizations squeezing budgets and economizing where possible, some businesses are reluctant to spend. During economically challenging times, many organizations opt to soldier on, using tools already in place (if any exist) and machines that are already at work in their current state. Money is not spent, so our costs are lower. What this "cost saving" doesn't address is the time trade-off made to perform these tasks. The Cost of Testing is not exclusively about money, very often we must include time in that cost analysis as well.

## Where Are We Spending Our Money?

As our development project ramps up, the resources available will dictate the total time needed to complete it. The number of people on a project certainly adds to the total price tag of a project, as well as the total amount of time a project may take. With a testing team, two testers working together will very likely complete more testing than one tester will. The combined test coverage is often much greater than the sum of individual tester's standalone efforts.



Having more powerful computers will certainly shave the time it takes to build applications. Likewise, having more machines makes it possible to deploy more test environments.

Support for multiple Operating Systems provides additional challenges. Microsoft Windows, MacOS, and HP-UX are examples of Operating Systems that have an up-front licensing cost for each installation. Linux and Free-BSD are examples of open source Operating Systems that are available for free. Beyond just the cost of the Operating Systems is the time needed to give each system thorough testing. Each supported platform adds a time multiplier as to how long it will take. There are also variations in Database software, web server software, Office Productivity applications, and other components that may be required to evaluate the functionality of the application under test. In addition there are variations in the way the software is written and deployed (pre-compiled code such as C++ vs. dynamic application code like Java). Each variation requires another environment be set up and tested.

Testing these multiple OS's requires choices be made. We can purchase and maintain individual hardware machines. We can also utilize virtualization technology to reduce the need for multiple physical computers. The host server will require many orders of magnitude more capacity and performance than the individual computers we would have virtualization replace. If a project requires six systems of equal capability, virtualizing those machines will require sufficient resources on a single server to be able to house and run those six environments. That host server will require enough disk space, RAM and system resources for the six virtualized environments to operate and respond as though they were standalone machines.

Testing tools range from free, open source options all the way up to very expensive proprietary commercial systems. The application under test often determines which tools, if any, will need to be used. There are many free and open source test tools for testing web applications. By contrast, most of the tools that test a compiled application from a "black box" perspective are commercial products (though there are also a number of open source options available that allow us to create tests and automate many tasks).

As I have seen over the years, not spending for these items or areas can provided a short-term cost savings. We made do with the servers and machines we had on hand. We also managed to get by with a limit on software licenses (within reason and within the bounds of what is legal; I do not advocate piracy to cut costs at any time). I have set up test beds and test labs using free OS software, using a variety of Linux variations. I have also used both commercial and free virtualization software, where the host was running Linux, and the guest machines likewise represented a variety of free operating system options. These test environments also have been set up to use as many free or inexpensive variations of open source tools for testing purposes. There is no question that using these techniques has saved money.

### **What's The Tradeoff? The Trade-Off Is Time**

There is an adage that says "You can have a system that is cheap, fast, or of good quality. Pick two." Price, time and quality rarely move up and down together. If one area gets incremented, others are decremented. When we spend less to perform a task, the time to accomplish the task often increases. The quality of the resulting output may also decrease. In my experience, companies want to bring costs down, while keeping the quality level as high as possible. To make both goals succeed (lowering costs and improving quality), in most cases we will have to allow time to increase.

Let's use a build server as an example. If we chain a number of computers together to perform builds the overall time to complete the build process will diminish. What could shaving 25% of the time off completing a build do for your development and testing teams? Think of the amount of time that could be applied to testing and fixing issues discovered, rather than waiting for the build to finish. The quicker the turnaround from development to testing, the quicker issues can be discovered, addressed and fixed.

Quicker builds allows for better integration of new code. Quicker builds allows for a system that is up and running and more frequently available. With the effort to set up the scripts and tests needed, the quality of the build process can be improved. The productivity gains were produced by spending money to add server horsepower. Paying for the human brainpower to create scripts and processes makes for fewer errors in the build process.

To contrast, let's consider what would happen if a new initiative of ours requires a server, but our budget doesn't allow for it. We could share the workload on the existing build servers. We could pull one of the machines out of the build process and make it a standalone server. What happens? Cost savings are immediate; we didn't have to purchase a new server. The quality of the builds may be un-affected, but the time to complete the builds has increased. If the system performs daily builds, the time we have lost because of the server change will be multiplied by the number of days for as long as the projects are active. The benefits of quicker turnaround for development and testing will not be achieved.

I have used test environments with open source operating system software and open source testing tools. On the surface, the environments appear to be totally free or have a very low physical cost to implement. In my experience, deploying, maintaining and administering these operating systems and applications requires a significant time commitment on the part of the development, test and IT teams. Open source software (and the communities that help with bug-fixes, documentation and implementation of new features) is on par with the development of many commercial applications. The ease of installation and maintenance of the installed software has likewise improved dramatically. Still, a high level of knowledge and skill is required to keep multiple open-source applications and tools running smoothly. That knowledge development and maintenance requires time.

Several groups I have been a part of, especially at the beginning, had no formal tools in place. Most of the testing steps are performed manually. Many tasks are best suited to active manual testing, but when all our testing is manual, the time required to complete it can be significant. Multiply our manual testing efforts by the number of projects. The time commitment to do all of the needed testing rises exponentially. Few organizations have the luxury of long periods of time to make sure all testing is performed. What tends to happen is that testing time gets cut (or at best held to the required scheduled time). The total amount of tests get cut, which can lead to a lower quality product. When we apply a risk based test approach, we aim to maximizing the potential of covering areas we consider most critical. Even with this method, consider how long it would take to cover all areas in the time limit. Where will the cuts be made? Should we eliminate human factors testing? Will considerations for user experience go untested or receive a lower priority? Will features that are desired but can't be fit into the timeline be moved to later releases? If so, how long will it be until we will be subject to the hands of Kronos once again?

*To be continued in next issue ...*

[Discuss this topic on QualityTesting](#)  
[Click HERE](#)

**Michael Larsen** is a lone tester in San Francisco, California.

He has spent seventeen years in testing and test organizations, ranging from network routers and web caching devices to distributed databases dealing with the legal and entertainment industries.

Michael is a Brown Belt in the Miagi-do School of Software Testing, an instructor with the Association for Software Testing, and the co-founder and facilitator of Weekend Testing Americas.

Michael blogs at [www.mkl-testhead.blogspot.com](http://www.mkl-testhead.blogspot.com) and can be found on twitter at @mkltesthead.





Plenty of software testing books tell you how to test well; this one tells you how to do it while decreasing your testing budget.

A series of essays written by some of the leading minds in software testing, **How to Reduce the Cost of Software Testing** provides tips, tactics, and techniques to help readers accelerate the testing process, improve the performance of the test teams, and lower costs.

20% OFF for  
our readers

# How to Reduce the Cost of Software Testing

Edited by Matthew Heusser and Govind Kulkarni

Cam Kanar  
Matt Heusser  
Serena Delisle  
Govind Kulkarni  
Catherine Powell  
Michael Larson  
Michael Burton  
Ed Barkley

Michael Kelly  
Jeroen Rosink  
Karen Johns  
Petteri Lyytinen  
David Gilbert  
Markus Gartner  
Justin Hunter • Gary Beck  
Curtis Stoelkenberg  
Scott Barber  
Matt Heusser  
Catherine Powell  
Jonathan Bach  
Anne-Marie Charrel

Foreword  
What Will This Cost Us?  
The Cost of Quality  
Testing Economics  
Opportunity Cost of Testing  
Trading Money for Time  
An Analysis of Costs in Software Testing  
Test Readiness: Be Ready to Test When the Software Is Ready to Be Tested  
Session-Based Test Management  
Postpone Costs to Next Release  
Cost Reduction through Reusable Test Assets  
You Can't Waste Money on a Defect That Isn't There  
A Nimble Test Plan: Removing the Cost of Overplanning  
Exploiting the Testing Bottleneck  
Design of Experiments-Based Test Case Design  
Reducing Costs by Increasing Test Craftsmanship  
Rightsizing the Cost of Testing  
Immediate Strategies to Reduce Test Cost  
25 Tips to Reduce Testing Cost Today  
Rapid Test Augmentation  
Cost of Starting Up a Test Team

 **CRC Press**  
Taylor & Francis Group  
AN AUBACH BOOK

Enter your Discount Code as

**813DA**

to avail 20% off on CRC Press.

**CLICK HERE** to purchase it right away.

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](mailto: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.





A photograph of several students in a classroom, seen from behind, with their hands raised in the air. They are facing a chalkboard that has some faint writing on it. The students are wearing colorful shirts: light blue, red, orange, and green. The entire image is framed by a thick black border.

# In the school of Testing

*for your better learning & sharing experience*



# Lean Test Case Design

*By Darren McMillan*



I used to dislike writing test cases a lot! I found it pain staking how much work was involved just to document what was in my head. I used to think this could be time better spent doing other things because I did waste a lot of time writing these, too much time, time that could have been spent testing!

So like everything I thought could be done better I started trying to think of ideas on how this could be improved. Obviously it starts out with a question "What is the actual problem here?" The problem being my dislike for writing test cases & my desire to test instead! So looking back now it was obvious that I felt burdened by writing test cases on functionality which I already had available to test. I probably felt that for every test case I'd written I could probably have found some defects on that testable functionality.

## **Dilemma solved!**

At the start of each new release I'd always have a bit of spare time which I could use to get other projects done. This time could have been spent writing my test cases right? If I had some requirements I could start writing down test conditions. So that was it, I'd solved my dilemma of having to write test cases while all I wanted to do was test, I'd just write them before the functionality was written.

## **Requirements review**

So the release went out & the requirements for the next release came in. This was my chance to see if this made me feel any better. So out came the requirements & on went the thinking cap. I began jotting down my test ideas & looking at what types of testing would be required for this project. I also began noticing shortcomings in the requirements & things I knew just wouldn't work. I also began to generate ideas! Why do we do this? Or would it not be more user-friendly to do it this way? On and on it went, I was having loads of fun.

So I looked over what I'd noted down after reviewing these requirements & noticed that half of it mapped out as an overview of how I'd test this functionality & the other half was good feedback for the BA's & managers to consider. I'd found that on one hand I could now easily write up my test plan's & work out how much time I needed to write up test cases. On the other I'd come away with a lot more useful feedback than I'd ever had before, some which might just make this project that little bit more successful if all went to plan. I was very pleased & most of all I was having great fun.

### Introducing test cases early

Next stop was the test cases I'd hated writing before, this time around though it proved to be very enjoyable. A few days later I'd written up all my test cases & while I knew they'd probably have to change a bit I was quite happy in that not only was it more enjoyable, as they were being written I'd been spotting lots more gaps & potential issues with the requirements. The BAs really appreciated the feedback & I was happy that I was more involved with them to.

So the release came and went, I'd begun my first steps into proactive testing without realizing that I was being proactive. I'd noticed a big downturn in defects but I'd also spent a lot of time chasing developers to run these test cases I'd written. This seemed like a better way of working, making good use of a quite period & contributing heavily to the team.

### Lean in all aspects

So your probably wondering when the lean part comes in right? We'll begin touching on it now I promise 😊

### But how many test cases?



Previously our entire team appeared to have their own ways of creating test cases with no set standard. Some people opted for a test case for every one to three test conditions, resulting in hundreds of test cases. Others would bunch more conditions into test cases but for each step in the process that changed they'd always have to write a new test case. I didn't think this was very efficient, or fair at the same time for those people who'd have to re-run them if they ever became part of the regression test suite. I remember once two testers responsible for an important new feature got asked to write some test cases for it & came back with over eight hundred! Crazy & looking into these each had one or two conditions at most for each test case. How many test cases you've written means nothing, [James Christie](#) wrote an excellent article on this "[But How Many Test Cases](#)". Could you ever imagine asking someone to run those eight hundred test cases with a straight face? Crazy talk!

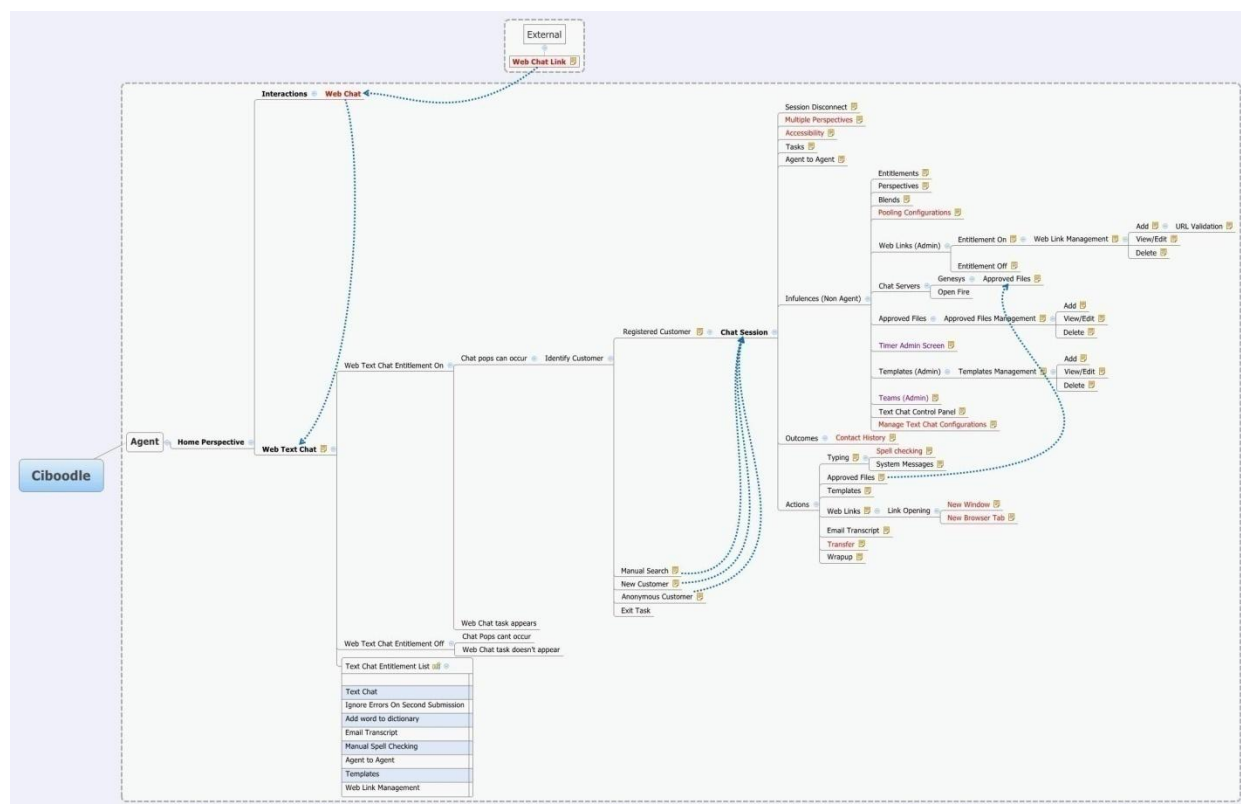
So that was the first challenge getting the rest of the team on board with how I'd thought these should be written. I'd arranged a meeting & asked that everyone submit how they thought a test case should look. We discussed the pros & cons of each template & grouped our favourite couple, then discussed these some more. Thankfully they all seemed bought into my suggested format.

### Mind mapping

So what was the format? We'll it's evolved slightly since then so we'll just discuss what it looks like at present. Firstly I think it's important to discuss how they are designed.

So like I'd said before I generally try to take a proactive approach by generating these up front. If you've paid any attention to requirements at the start of a project you'll know they can vary greatly from someone's vision to very detailed documentation. So you'll know the extent of your test cases will be limited by the available requirements, something which we've been lucky with at my company allowing us to provide very good test cases up front for developers to run prior to committing code.

Once I've gave feedback on the requirements & they'd become more stable I'd crack open my favourite mind map tool, in this case **Xmind**. Mind maps are excellent, previously I'd been writing all my test cases in our test management tool & not seeing as much interlink between the different area's of the feature as my thoughts were constrained to my current test case. I'd also have to jump in and out of test cases whenever I'd thought of new conditions for a test case I wasn't currently creating or editing. With a mind map as you build it up you just see all the links in the feature so you're fully aware of its integration points & influences.



With Xmind you can add extensive notes to each node on the mind map, this is what I used to write up my entire test conditions for that part of the feature, to later by dump into the test management tool.

If I think of a new test condition for some other part of the feature I simply select and edit the notes of that node on the mind map. What took a little time before now takes seconds. Likewise as everything is visibly in front of me my mind begins to naturally consider more aspects of the feature in turn generating better test conditions that I might previously not have thought of when using a test management tool to write these.



## Monkey Madness!

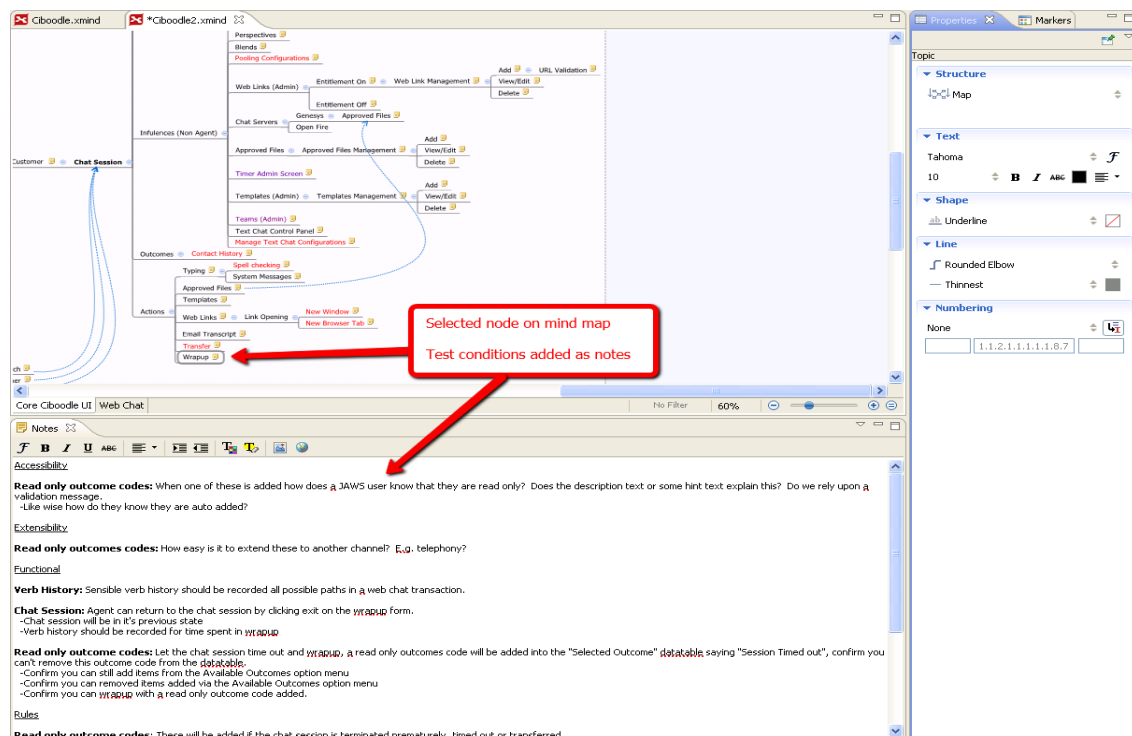


For me my map shows my path through the application/feature, or steps as you'd call them in a test case. When it comes a time to put my conditions into a test management tool the map becomes the test case folder hierarchy, making it easier to navigate test cases when running them in the future. If my tool needed me to add in steps for some reason (mine doesn't & I don't) I'd simply copy the folder structure onto my steps in a brief format e.g. Make Order > Add Billing Details > Confirm. Let's keep the fluff to a minimum that's why I myself don't use steps, people are intelligent & in my app can see that the folder structure mimics the steps they would have to take.

As you'll imagine with the nodes being paths or areas through the application or feature we can cover a broader feature area than before. We're not training monkeys by telling them "for this condition you'll need to select X, then after a second Y will appear" but to do that you need another step 🤪 oh crap lets write another test case for this! No we don't need to hold people hands; we can let people think for themselves. As long as the test conditions make sense people will understand that they might need to deviate from the beaten path to test something.

## Adding in types of testing

For the conditions I generate these on nodes on my mind map. I split them up into the types of testing I would do. When I say a testing type it would be something like Usability, Extensibility, and Security and so on. I generally dump a generic template of all testing types into each node at the start as a note so I can at least attempt to consider if any conditions need to exist for that testing type. From splitting these test conditions into different types of testing, you'll find that you think more, as such generating much better test cases for that functional area, since you're paying much more attention to what you'll need to test. I also add in a rules section for everything that the requirements expect, I guess you could call these our acceptance conditions.



## Increased requirements review

My final section at the end of my notes is for questions I might have about the requirements. The good thing about Xmind here is that if I know something on my mind map has some unanswered questions noted, I can highlight it in a different colour such as red. Then I can easily see that I need to resolve a few questions for that area at some point. You'll find when writing test cases up front even if you've done a review & feedback of the requirements you'll still find lots of other things you hadn't considered that will need some discussion with a stakeholder at some point. These could be gaps, risks or suggested improvements; you'll see it's very handy to be able to collate everything you've done in one place with a mind map.

## Generating the test cases

So you might be thinking these are all very conditional based, your correct! That's all I want. I'll take these once I'm happy with them from the mind map & put them directly into my test management tool as-is. Like I said my steps will be my folder hierarchy or dependant on your tool a very brief copy of the mind maps path to the node to replicate the steps the user would make. If I need some setup tasks done before running these test cases I'll put a link to them at the start of my conditions.

The screenshot shows the SwordCiboodle TestLink 1.8.1 interface. The left pane displays a tree structure of test cases, including 'Text Chat Entitlement On(47)', 'Identify Customer(45)', 'Chat Session(41)', 'Actions(11)', 'Influences (Non Agent)(25)', and 'Outcomes(1)'. The right pane shows the details for test case 'Cib-3717:Wrapup', including a summary table with 'Steps' and 'Expected Results'.

Annotations with red arrows point to specific parts of the interface:

- Grouping by the type of testing means the user can be more open to thinking of other tests while running these** (points to the 'Actions(11)' folder in the tree)
- Lots of conditions for a feature area = less bloat** (points to the 'Influences (Non Agent)(25)' folder in the tree)
- The steps are our tree structure** (points to the 'Steps' column in the summary table)

The summary table for 'Cib-3717:Wrapup' includes the following sections:

- Accessibility**  
**Read only outcome codes:** When one of these is added how does a JAWS user know that they are read only? Does the description text or some hint text explain this? Do we rely upon a validation message?  
-Like wise how do they know they are auto added?
- Extensibility**  
**Read only outcomes codes:** How easy is it to extend these to another channel? E.g. telephony?
- Functional**  
**Verb History:** Sensible verb history should be recorded all possible paths in a web chat transaction.  
**Chat Session:** Agent can return to the chat session by clicking exit on the wrapup form.  
-Chat session will be in it's previous state  
-Verb history should be recorded for time spent in wrapup  
**Read only outcome codes:** Let the chat session time out and wrapup, a read only outcomes code will be added into the "Selected Outcome" datatable saying "Session Timed out", confirm you can't remove this outcome code from the datatable.  
-Confirm you can still add items from the Available Outcomes option menu  
-Confirm you can removed items added via the Available

Requirements change they always do & often are not strictly followed, that's why I tend to leave all my test conditions in my mind map right up until the last minute. It's easier to change & add conditions in the map, it's less easier to do so once they've reached your test management tool. With the addition of a better testing mindset using the mind map you'll be more aware of the impact of a requirements change, as such you'll be able to write better conditions around this.

I'd considered doing a demonstration of all this using a simple application but I think I've covered almost everything & hopefully made sense. If anyone would like me to do a demonstration from requirements, to maps, through to the end test cases I'd be happy to, just leave a comment or drop me an email & I'll do this as a later follow up this article.

## Summary

So I hope from the description & the screenshots I've provided you'll be able to see the benefit of keeping your test cases as lean as possible. For a quick recap here's the why:

- Mind mapping
  - Increases creativity
  - Reduces test case creation time
  - Increases visibility of the bigger picture
  - Very flexible to changing requirements
  - Can highlight areas of concern (or be marked for a follow up to any questions).
- Grouping conditions into types of testing
  - Generate much better test conditions
  - Provides more coverage
  - Using templates of testing types makes you at least consider that type of testing, when writing conditions.
  - When re-run these often result in new conditions being added & defects found due to the increased awareness
- Lean test cases
  - Easy to dump from the map into a test management tool
  - If available the folder hierarchy can become your steps
  - Blend in easily with exploratory testing. Prevents a script monkey mentality.
  - Much lower cost to generate and maintain, whilst yielding better results.

That's it for this article! I hope you enjoyed it as much as I did writing it, thanks for reading.



**Darren McMillan** has been working in the testing field for over four years now. He has a genuine passion for all things testing and actively seeks to solve the problems others tend to accept. He strongly believes that opportunities are there to be taken and actively promotes self learning to others. When he is not testing or writing about his experiences, he enjoys nothing more than some quite family time.

A proud father to a beautiful daughter he hopes that from leading by example he will encourage her to follow her own dreams. Contact Darren on Twitter @darren\_mcmillan.





are you one of those  
#smart testers who  
know d taste of #real  
testing magazine...?



then you must be telling your friends about ..



Tea-time with Testers

Don't you ? 😊



Tea-time with Testers !

first choice of every #smart tester !





# Testing...



# with Anurag

## Testing Checklist for Mobile Applications

### 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**

For any tester who is new to Mobile apps testing, it often becomes tough to ensure that he/she has tested the application in all aspects.

Most of the times testers tend to miss important aspects of any application or say its features based on different criteria. Also, even though one is experienced enough to test particular mobile app in maximum possible aspects, it is tough to say that he/she will be able to test another app with same cleverness.

Here is the checklist that I have prepared based on my experience which will help you to ensure that your application under test is tested thoroughly.

No.	Module	Sub-Module	Test Case Description	Expected Result
1	Installation		Verify that application can be Installed Successfully.	Application should be able to install successfully.
2	Uninstallation		Verify that application can be uninstalled successfully.	User should be able to uninstall the application successfully.
3	Network Test Cases		Verify the behavior of application when there is Network problem and user is performing operations for data call.	User should get proper error message like "Network error. Please try after some time"
4			Verify that user is able to establish data call when Network is back in action.	User should be able to establish data call when Network is back in action.
5	Voice Call Handling	Call Accept	Verify that user can accept Voice call at the time when application is running and can resume back in application from the same point.	User should be able to accept Voice call at the time when application is running and can resume back in application from the same point.
6		Call Rejection	Verify that user can reject the Voice call at the time when application is running and can resume back in application from the same point.	User should be able to reject the Voice call at the time when application is running and can resume back in application from the same point.
7		Call Establish	Verify that user can establish a Voice call in case when application data call is running in background.	User should be able to establish a Voice call in case when application data call is running in background.
8	SMS Handling		Verify that user can get SMS alert when application is running.	User should be able to get SMS alert when application is running.
9			Verify that user can resume back from the same point after reading the SMS.	User should be able to resume back from the same point after reading the SMS.
10	Unmapped keys		Verify that unmapped keys are not working on any screen of application.	Unmapped keys should not work on any screen of application.
11	Application Logo		Verify that application logo with Application Name is present in application manager and user can select it.	Application logo with Application name should be present in application manager and user can select it.
12	Splash		Verify that when user selects application logo in application manager splash is displayed.	When user selects application logo in application manager splash should be displayed.
13			Note that Splash do not remain for fore than 3 seconds.	Splash should not remain for fore than 3 seconds.
14	Low Memory		Verify that application displays proper error message when device memory is low and exits gracefully from the situation.	Application should display proper error message when device memory is low and exits gracefully from the situation.
15	Clear Key		Verify that clear key should navigate the user to previous screen.	Clear key should navigate the user to previous screen.
16	End Key		Verify that End Key should navigate the user to native OEM screen.	End Key should navigate the user to native OEM screen.
17	Visual Feedback		Verify that there is visual feedback when response to any action takes more than 3 seconds.	There should be visual feedback given when response time for any action is more than 3 second.
18	Continual Keypad Entry		Verify that continual key pad entry do not cause any problem.	Continual key pad entry should not cause any problem in application.
19	Exit Application		Verify that user is able to exit from application with every form of exit modes like Flap,Slider,End Key or Exit option in application and from any point.	User should be able to exit with every form of exit modes like Flap,Slider,End Key or Exit option in application and from any point.
20	Charger Effect		Verify that when application is running then inserting and removing charger do not cause any problem and proper message is displayed when charger is inserted in device.	When application is running then inserting and removing charger should not cause any problem and proper message should be displayed when charger is inserted in device.



21	Low Battery		Verify that when application is running and battery is low then proper message is displayed to the user.	When application is running and battery is low then proper message is displayed to the user telling user that battery is low.
22	Removal of Battery		Verify that removal of battery at the time of application data call is going on do not cause interruption and data call is completed after battery is inserted back in the device.	Removal of battery at the time of application data call is going on should not cause interruption and data call should be completed after battery is inserted back in the device.
23	Battery Consumption		Verify that application does not consume battery excessively.	The application should not consume battery excessively.
24	Application Start/Restart		1. Find the application icon and select it 2. "Press a button" on the device to launch the app. 3. Observe the application launch in the timeline defined	Application must not take more than 25s to start.
25	Application Side Effects		Make sure that your application is not causing other applications of device to hamper.	Installed application should not cause other applications of device to hamper.
26	External incoming communication – infrared		Application should gracefully handle the condition when incoming communication is made via Infra Red [Send a file using Infrared (if applicable) to the device application presents the user]	When the incoming communication enters the device the application must at least respect one of the following: a) Go into pause state, after the user exits the communication, the application presents the user with a continue option or is continued automatically from the point it was suspended at b) Give a visual or audible notification The application must not crash or hung

Feel free to drop me a note if it helped you or you need more information/help around it.

[Back To Index](#) 

Have a great day and wish you all a very Happy Diwali !

***Wish you ...***



***HAPPY DIWALI***

***from Tea-time with Testers family***

# testing intelligence

- *its all about becoming an intelligent tester*



an exclusive series by **Joel Montvelisky**

## What do you pack when you go for a Bug Hunt?

Bug Hunting is becoming a common practice form many testing organizations world wide; yet some test managers wrongly feel they go hunting when their testers informally play with the application in order to find "border-case defects".

Bug Hunts are Informal Testing exercises; this should not be mistaken with playing with the system without a purpose or objective. In order to achieve something (and not waste your time!!) during these activities you need to follow a specific methodology, perform planning and preparation actions, and monitor and control the process throughout its execution.

Even if there is no written book (at least that I am aware) on Bug Hunts, I follow a process that I caught some years ago during one of the STAR conferences.



I plan Hunts based on *Pair-Testing* and *Soap Opera Scenarios*, combined as a *tournament* between teams.

**Pair Testing** is when you take 2 engineers and they work as a team on 1 computer (or environment). The idea comes from **pair programming** with all its known advantages such as knowledge sharing, brain storming, and positive pressure from working with a partner. I've had very good results pairing engineers from different teams; especially when taking development engineers and pairing them with test engineers, leveraging the advantages and points of view from both sides of the process.



**Soap Opera Scenarios** are relatively short end-to-end scenarios that take the system from beginning to end of a complex operation on a fast (and sometimes exaggerated) chain of events. Working under extreme conditions we are able to find issues that we missed during our controlled scripts and scenarios.



**The Tournament** activity is where the human factor comes into play. You want people to be motivated, and what can be a better motivation for an engineer than a prize..? I keep track of many statistics like the number of original bugs detected per day, the most serious bug detected, the worst system crash, etc; and give a prize to the pair who exceed on each category (we usually give away t-shirts and on one extreme case we even gave an iPod!) .

In any case it is not the prize but the spirit that makes the difference.


There are some rules of thumb I give QA organizations before they start planning their Bug Hunts:

1. Make sure you've reached a minimal *application maturity level*; if it is still too easy to detect critical bugs or the system keeps crashing all the time you may be ahead of your time.
2. Work on testing environments with *real life data*. Your tests need to be far from sterile in order to simulate a real working environment.
3. Create a *balanced activity schedule*. I work based on 2-day bug hunts, where each day starts with 4 hours of *Exploratory Testing* around a specific application area (each team or two working on a different part of the application); and during the second half of the day we run our Soap Opera scenarios, where each team receives a user profile and a list of high level tasks.
4. Manage the activity using a *tracking system*. Any bug tracking system will help you keep track of the issues detected by the teams; if possible use a *dashboard* to make sure all team members are updated on the position of the rest of their peers.
5. Have a *bug referee*, she will decide whether the findings are real bugs and also stop duplications from been reported and counted.
6. Create *anticipation* by having internal teasers and "publicity campaigns" ahead of the activity.



7. During the hunt generate an *informal atmosphere* by playing music, giving each team bells to signal whenever they find and report a new bug, bringing pizzas and sodas, and using other out-of-the-ordinary things that will create the feeling of a special occasion.

Bug Hunts are as much about the right atmosphere and state-of-mind as they are about the methodology and scenarios you run. Make sure you and your team have fun and it will surely generate the outputs you expect it to produce or more.



**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](#)

Back To Index 

# Teach-Testing →

An Ambitious Campaign by Tea-time with Testers

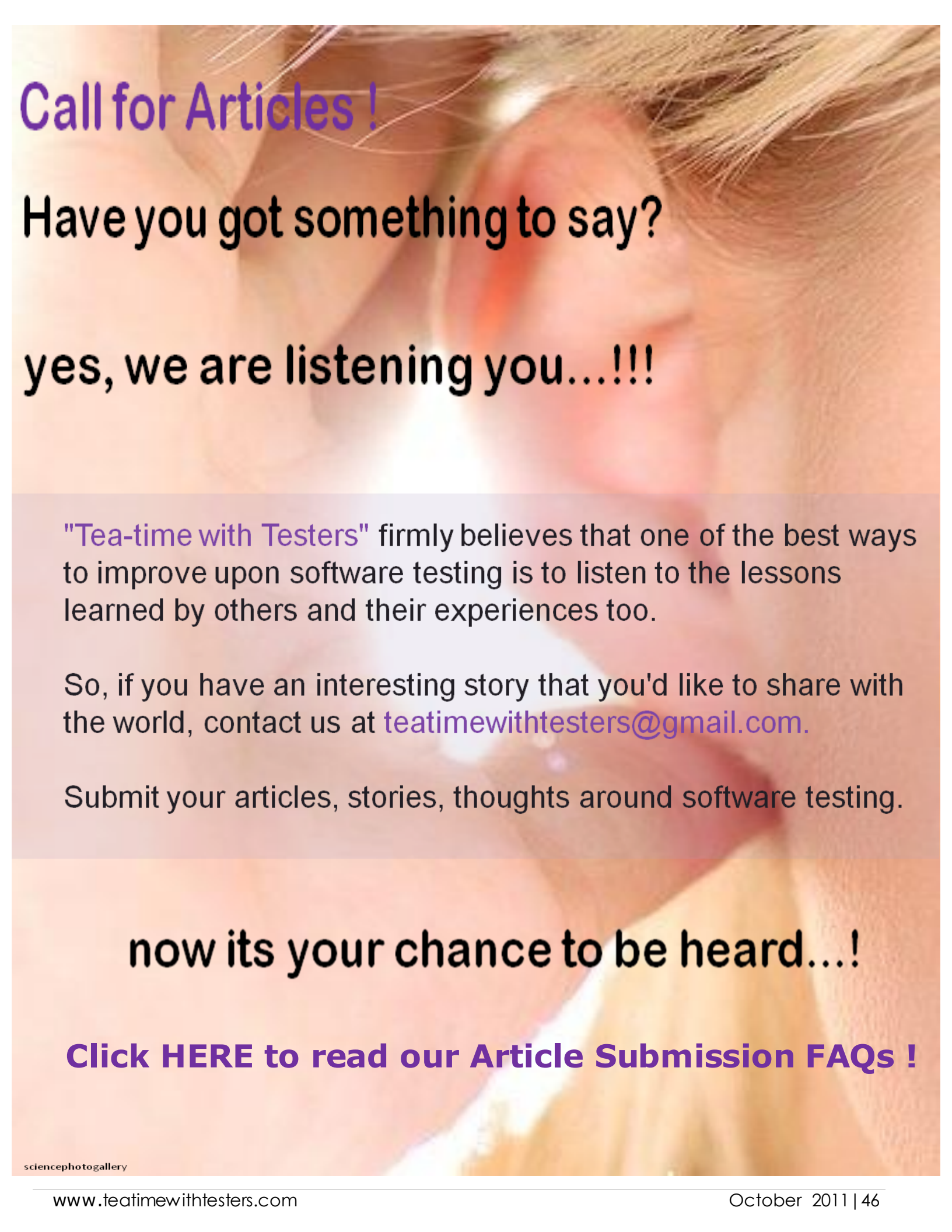


Click here to Cast Your Vote



by





# 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](mailto:teatimewithtesters@gmail.com).

Submit your articles, stories, thoughts around software testing.

## now its your chance to be heard...!

## Click **HERE** to read our Article Submission FAQs !

# T ' Talks



*T. Ashok exclusively on software testing*

## **Seven consecutive errors = A Catastrophe**

"A typical accident takes seven consecutive errors" quoted Malcolm Gladwell in his book "The Outliers". As always Malcolm's books are a fascinating read.

In the chapter on "The theory of plane crashes", he analyses the airplane disasters and states that it is a series of small errors that results in a catastrophe. "Plane crashes are much more likely to be a result of an accumulation of minor difficulties and seemingly trivial malfunctions" says Gladwell.

The other example he quotes is the famous accident - "Three Mile Island" (nuclear station disaster in 1979). It came near meltdown, the result of seven consecutive errors -

- (1) blockage in a giant water filter causes
- (2) moisture to leak into plant's air system
- (3) inadvertently trips two valves
- (4) shuts down flow of cold water into generator
- (5) backup system's cooling valves are closed - a human mistake
- (6) indicator in the control room showing that they are closed is blocked by a repair tag
- (7) another backup system, a relief valve was not working.



This notion is reflected in the book "Ubiquity" by Mark Buchanan too. He states that systems have a natural tendency to organize themselves into what is called the "critical state" in what Buchanan states as the "knife-edge of stability". When the system reaches the "critical state", all it takes is a small nudge to create a catastrophe.

Now as a person interested in breaking software and uncovering defects, I am curious to understand how I can test better. How do you ensure that potential critical failures lurking in systems that have matured can still be uncovered?

Let us look at what we do- We stimulate the system with inputs (correct & erroneous) so that we can irritate latent faults so that they may propagate resulting in failure. When the system is "young", the test cases we design are focused on uncovering a specific singular faults i.e. a set of inputs that can irritate singular faults and yield possibly critical failures. This is possible because the "young system" is not yet resilient and therefore even a singular fault bumps it up! We then think that our test cases (i.e. combinations of inputs) are powerful/effective. But these test cases do not yield defects later as the system becomes resilient to singular faults.

As the system matures we need to sharpen the test cases to irritate a set of potential faults that are tied to create a domino effect to yield critical failures. Creating test cases to uncover singular faults in a mature system is no more useful. It is necessary that test cases be at a higher level of system validation (i.e. have long flows) and have the power to irritate a set of faults.

So can we resort to uncovering critical failures only by testing by creating test cases at higher levels that have the power to uncover multiple types of faults? Not necessarily. We can apply this thought process at the earlier of design/code too. Using the notion of sequence of errors and understanding what can happen.

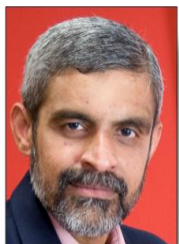
If your drive in India you know what I mean ... the potential accident due to a dog chasing a cow, which is charging into the guy driving the motorbike, who is talking on the cell phone, driving on the wrong side of road, encounters a "speed bump" , and screech \*@^%... You avoid him if you are a defensive driver.

Alas we do not always apply the same defensive logic to other disciplines like software engineering commonly enough...!

Have a safe day!

Twitter @ash\_thiru

[Back To Index](#)



## Biography

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

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

He can be reached at [ash@stagsoftware.com](mailto:ash@stagsoftware.com) .





## OUR PARTNERS

## 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.



*Continued from page 3 ....*

We were programmers. The sysadmin was a programmer, the guy who built the computers was a programmer ... the company president wrote the main code for the initial product offering.

This wasn't a programming company; it was a *company of programmers*.

And, without any testers, analysts, user interface people, graphic designers, configuration managers, or anything even remotely like that on staff, we were programming without a safety net.

Keep in mind, I was twenty years old and straight out of school.

Stuff went wrong. It was kind of a humbling time.

I did, however, quickly come to a realization of my ignorance -- that we needed some sort of safety net; that the company was not going to provide it, so I had better build it myself. I decided to do something about it, going back to school. At first I was taking interesting classes at night while vaguely working on MBA pre-requisites, but I eventually decided to pursue a master's degree in Computer Information Systems, a sort of mix between Computer Science and Business, at Grand Valley State University.

### **To Graduate School And Beyond**

We could debate school all day long. The long and short of it was that yes, I could have learned those things on my own, but the program pushed me to study specific things on a deadline. It provided me with enough exposure to key topics to know where to go to learn more, and, most important, it developed in me the habit of working in my own time on professional development, especially writing.

In the summer of 2003, I attended the O'Reilly Open Source Convention, and presented a lightning talk on Test Driven Development (TDD). That same summer I published my first article (outside of the school paper), entitled *How To Survive The Coming Bust*. The following spring, I attended STAREast, at the time the world's largest software test event, where I presented a track talk on Test Driven Development, and wrote an article for publication from my hotel room.

It was at one of those early STARS, that I first met Cem Kaner, and later James Bach. I remember how, at the Open Source Convention, the cool kids all had a "posse", a small group of followers. Except for the ones selling training services, I got the impression, that unless you'd written some impressive (well-known) (public) (open source) software; they didn't really want to talk to you.

Then I met Dr. Kaner. He recognized my name because I had posted on some internet forums. We talked for perhaps twenty minutes. I felt like I was abusing his time, and asked if I was keeping him from anything. "Yes, you are keeping me from my dinner. Would you like to join me?"

It felt like ... home.

Within a year or so of that meeting, I had a business card that said "Software Developer", but I was self-identifying as a tester.

Oh yeah. I was also writing.

### **And Then Life Got Busy (The Next Half-Dozen Years)**

It's been truly an amazing past few years of my life. I've been back to STAR Conference four times, presented at the STPConference four times, been out to present at Google, contributed a chapter to *Beautiful Testing*, the forward to *The Clean Coder*, and, most recently, served as lead editor for the anthology on *How To Reduce the Cost of Software Testing*. I've also been blogging since 2006, current at *Creative Chaos*, the *STPCommunity Blog*, and *IT Knowledge Exchange*. Oh, and I've written a fair number of other places, too. Plus there's the podcast; in my spare time I also served as lead organizer and first master of ceremonies for the *Great Lakes Software Excellence Conference*, facilitates liaison for the Conference for the *Association for Software Testing* in 2010, and, based on that, was elected to serve on the *AST Board of directors* for a 2011-2013 term.

And, yes, I had the day job the whole time, finally going independent in May of 2011. On my current assignment, I'm doing a full-time load as a consulting software tester forty hours a week, keeping my existing writing and volunteer commitments, and, yes, finding time to write for **Tea-time with Testers**.



The point of all this wasn't to bore you with my resume -- it was to reveal a few interesting truths about how I got where I am, that, I hope, might be helpful. If I may, I would like to provide a small piece of advice.

### Tomorrow

So there you have it. It's been a wonderful, long and winding road, with a few surprises, a few challenges and opportunities. I've tried to hit the highlights, especially my early career; in the way that I thought might be most helpful to you.

It might be a stretch, a bit of exaggeration, perhaps, to say I'm in love with Software Testing - but it's not a stretch by much.

You might just say I'm hooked.

### But What about You?

First, I'd like to remind you that I am a screw-up. As a cadet in Civil Air Patrol, I consistently failed. My attitude was poor; there was always some other cadet who could run faster, or had a better uniform, stood up more straight, or told the brass what they wanted to hear. The one thing I had was *perseverance*. The fast guy, the polished guy, the driller and the slick one all went on to other things, but I stuck around and kept testing and advancing. Toward the end, I even managed to have a few responsible assignments, but even then I managed to fail a bit.

Remember, I was a failure in ROTC too; I never did earn that commission.

As a college student, my grades were mediocre. I thought that many of my professors didn't like me; at least, they always seemed to be mad at me. Years later, after Beautiful Testing came out, one of my old professors explained they were *disappointed* -- they thought I had potential and I was wasting it.

I woke up one day, and I was a college graduate. And my momma and daddy weren't going to provide for my living anymore.

It was time to grow up.

I do not credit myself for this decision, but instead a still small voice encouraging me to do better, to rise above my own sloth and take responsibility for my life.

I credit the Holy Spirit of the Bible for taking the mess of a boy that came to Jesus Christ in 1995 and turning him into a man, and I'd be happy to talk about it some time. But, as they say ... that is another story. Shortly after that morning, well, things got better.

Somewhere along the line (probably during graduate school) I become both an idea guy and a finisher. That is to say, I have a few ideas ("lightning talks! podcasts! Let's run a conference! Let's write a book!") and I *actually do the work*. By that I mean actually doing software testing and helping others do it, while writing about it.

It turns out that perseverance trumps flash -- although having ideas helps.

My second piece of advice builds onto the first: don't let your mistakes define you. Instead, learn what not to do. Stop doing it. Then start over doing it right. It is not too late. Three strikes and you are *not* out.

Third, if you feel a little confused about your career options, don't fret too much. That's normal. Think about Matt the teen-ager; I was totally misguided on my future. I started out wanting to write fiction, played army for half a decade, programmer for years after that, and finally found software testing, my home. (But ask me again in another decade or so.)

My fourth piece of advice lines up with this: Experiment with a lot of things. Flow toward the things you actually enjoy that provide an outcome you are happy with.

We gave up on Big Design Up Front for software years ago. It probably won't work for your career either. It's good to come up with a theory about how your career should go. Actually testing it and adapting?

That's better!

I hope you enjoyed this issue of *Tea-time with Testers*, and I hope it helps you on your journey.

Sincerely Yours,

*Matt Heusser*

**Matt Heusser**



# Over a Cup of Tea

with

Matt Heusser



Matthew Heusser, a techie from Allegan, Michigan, travelled through a long and winding road to create his own identity. Today he is one of the leading personalities in software testing industry.

But what has he got to say over questions asked by TTWT?

Find it out in this interview.

Tea-time with Matthew Heusser !





Our readers now already know about your career as a Tester. Still, if asked to tell about your "love with testing" in few lines, what would your answer be?

How many jobs in the world can you say "this is never going to work because...", give some evidence, and collect both a pay check and a pat on the back? (Laughs)

One of the things I like the most about testing is how it presents new intellectual challenges every morning. Here we have this impossible problem: To know the state of the software, when the combination of inputs and outputs is infinite, yet we need to come to the conclusion in finite time.

Then you approve, get a new build, wake up the next morning and do it all over again - with totally different features and a new risk profile. How could you not enjoy this?

You have rich experience of working in various fields. Did you ever find testing as a monotonous job? What keeps you motivated to test better?

Usually I find that testing is one of those things where, if it isn't fun, you're doing it wrong. However, sometimes, I have to admit, I get tester fatigue. Usually it's something like this: an install is failing, or I don't see the change, so I have to uninstall and re-install and check, and then it doesn't work. Is it hitting the right database? Better check the config file. Re-run. etc. Generally these are most painful to me when I can't figure out what is wrong or the problem involves a long feedback cycle. You know, you set up a test, walk away, get a cup of coffee, and ... it's still wrong. So you set it up again ...

The fix for this is usually either to tighten the feedback loop, take a break, work on something else, or handoff the work. Once my brain is fried, I make mistakes, and the process just gets worse.

You are one of the established speakers in the industry. How important do you find conferences and other knowledge sharing initiatives to an individual and the community?

If you've spent most of your career at one company, you may not realize how screwed up you are, and make the same mistakes over and over again.

Note: I say that with confidence because we are all screwed up in different ways.

The next problem you'll have is that you do know how screwed up your company is. This is a real bummer. You can go to conferences to find out how screwed up everyone else is -- it'll make you feel better.

Eventually you realize that picking a job or picking a place to work, or picking what battles to fight, is all a matter of tradeoffs and values. Then you need people to bounce ideas off of.

So, while you don't particularly need to go to conferences, I do suggest some sort of meetup with other people in testing and software outside of your field. The conferences just give you a week away to really think about the problems without the busy business of production.



What difference does it make working in an organization and working as a freelancer? Why would someone go independent?

Now there's a long question !

Basically, I find the freelance life better in every way but one - predictability of income. If you need predictability of income, well, maybe a day job is for you.

If, on the other hand, you have a spouse that works outside the home, or you have about a year's worth of savings saved up, or you have some other form of generating revenue that can provide enough if you need it, I recommend going independent. Personally I built a modest writing/speaking business to be big enough, and put enough in the bank, to survive a reasonable down period. Considering that I had a day job, this took a fair bit of personal sacrifice. If the sacrifice isn't worth it to you, well, then, it isn't worth it to you!

Thinking on it some more, of course there are other variables. Unless you live in a major metropolitan area, you may have to travel to get freelance assignments, and that may be more (or less) acceptable to you and your family. For many people that is a deal-breaker - especially folks who want to actually do the testing work, instead of training or consulting. (With training and consulting, you can raise your rates a bit and be home a *little* more. )

Also note that unless they can grab a major, multi-year contract, independents are always in sales mode. If you don't like telling people 'No', and you don't like rejection, well, maybe being in sales mode all the time isn't the best bet for you professionally

'How to reduce the cost of Software Testing': How do you feel about it now ? Any special experience, memories, mentions ?

The process of developing the book was amazing! I got to meet, and really get to know, some serious thinkers and doers through the creation of that book. In October 2010, about the time we were finishing the second drafts, a group of us did a panel on that very subject at STPCon in Las Vegas, Nevada, USA. It came together very well, I think in part because at that point we really knew each other well.

Of course I have some regrets. Most of the contributors are not professional authors, and while I might qualify as a professional author, I'm a far sight from a professional editor.

Perhaps I'm too tough on myself. For what it claims to do, I was very pleased.







### What message would you like to send to our testing community?

Go get good at testing. Actually add real value on projects, and a lot of the fear and uncertainty and second-class-ness you feel, or that other people like to inject upon you, will sort of go away.

At the same time, get good at explaining testing - how you do it and why you do it.

There are plenty of bad ideas and mythology in the world of software testing. We need to let experience and reason win out over legend and dogma. The way to do that is for each of us to get very good at understanding what is really happening on projects, and also in explaining it to others.

Those are communication skills - things like writing and speaking. We don't talk about that very much in the testing world; we should.

### What do you think is the responsibility of the leaders/mentors in the field of software testing today?

I suppose it depends on what you mean by leaders and mentors. Personally, I think a large number of people just want to do what they are told - they want to follow. And, to some extent, that's okay -- as long as they are following the right people!

If a great number of testers are sort of sheep, then I want the leaders and mentors to be the sheep dogs; to protect the people who aren't yet able to protect themselves.

Along the way, we provide those people with the tools to help themselves.

So basically I'm suggesting you be a sheepdog who runs an ammo factory and basic training program or something. It's late, and I'm pretty tired. :-)

### Your opinion about Tea-time with Testers. Any suggestions?

I appreciate what you folks do for the community and I was pleased to be invited to contribute. I am amazed at the value you manage to provide the community for so little cost.

You should charge! And Thank you!

Thanks Matt! It was great fun talking with you.







# Service Test

## **PART 1**

***By Karthik Subramanian***

### **Overview of Service Test**

Service Test is Hewlett Packard's tool for automated testing of Web Services and other UI-less systems. It uses a visual designer as a canvas for creating steps in a test.

You create tests by dragging activities from the Toolbox onto the canvas and defining their parameters.

Service Test 11.0 has been totally redesigned over the previous version. In the previous versions of Service Test 9.x and below was based on the LoadRunner VUGen engine, the current version of Service Test has been totally redesigned from the ground up based on the Visual Test Designer Interface.

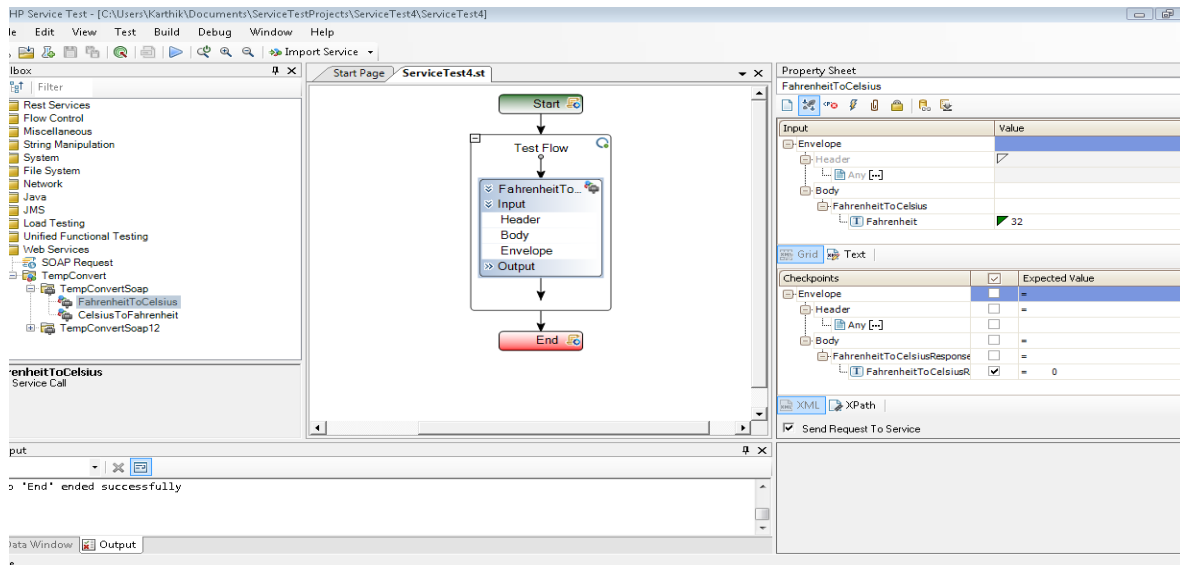
Also, currently HP is shipping Service Test as part of its Unified Functional Testing package along with QuickTest Professional (QTP), so that users can use QTP for testing GUI application and Service Test for testing Web Services and non-GUI applications.

### **Service Test GUI Interface**

The Service Test interface lets you design a test by dragging activities into a canvas.

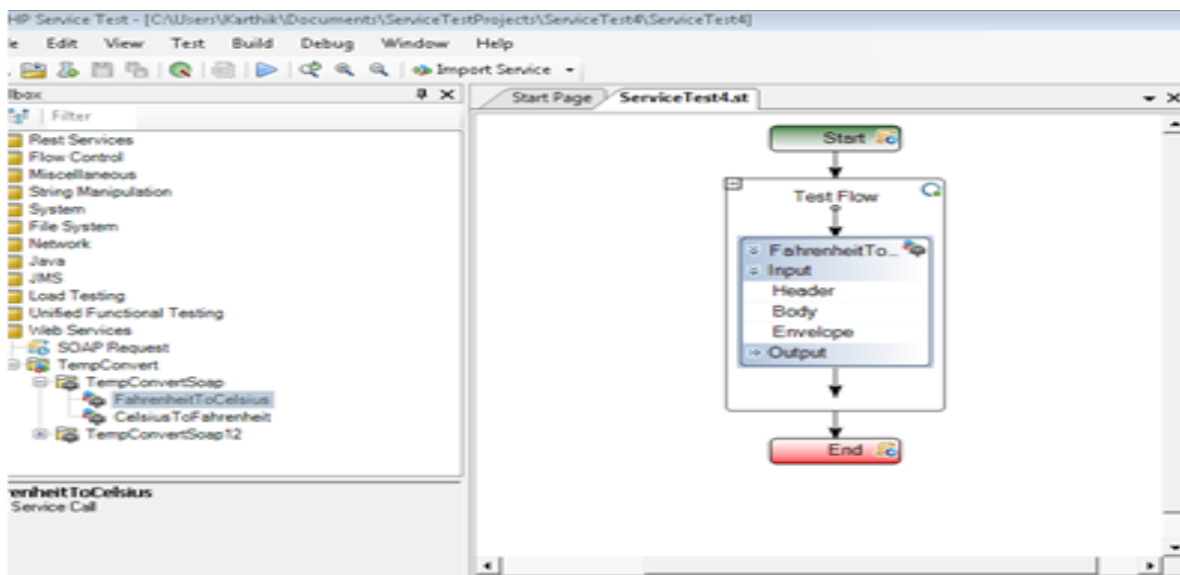
## Service Test Main Window

This screen is the Service Test's main window for creating and populating tests. It contains the toolbox palette, canvas palette, property sheet, output and error tab.



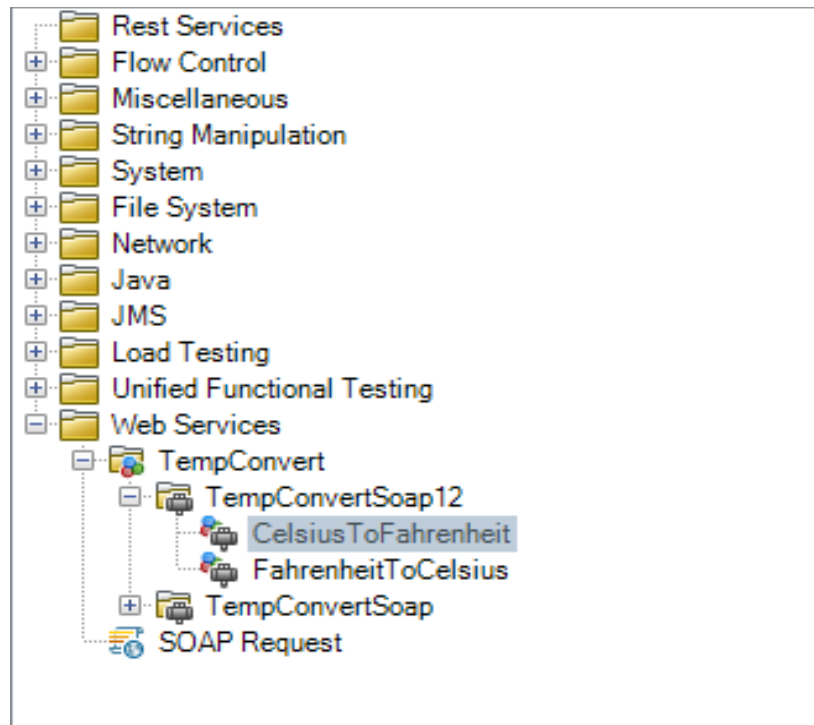
## Tests Pane

The Tests pane enables you to view all the test components in a tree hierarchy.



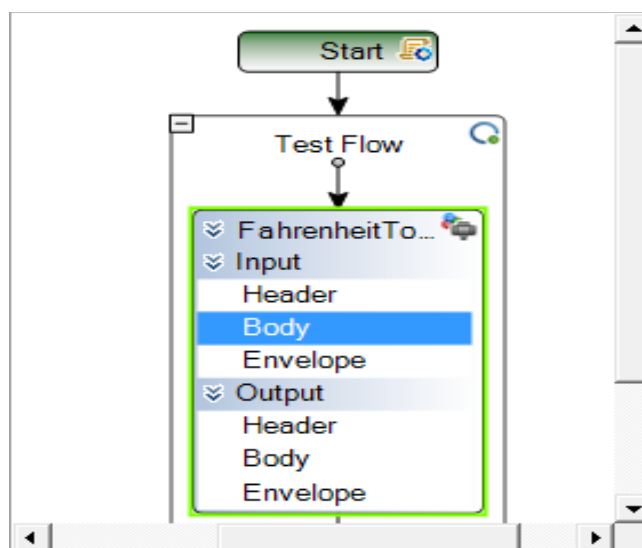
## Toolbox Palette

The Toolbox palette enables you to view all of the available activities. It contains all of the activities and flow control objects that you need in order to create a test.



## Canvas Palette

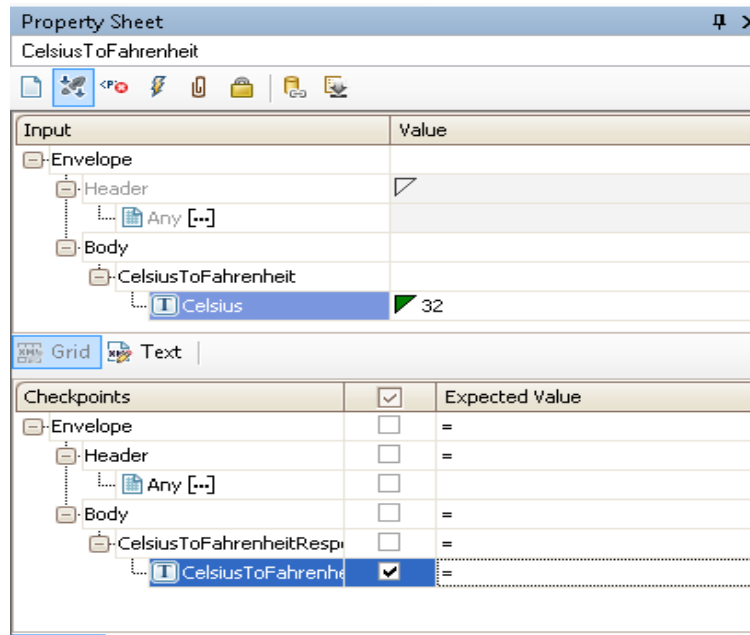
The Canvas palette enables you to drag and drop items from Toolbox. It provides a visual representation of the test flow. You populate the canvas by dragging in activities from the Toolbox palette.





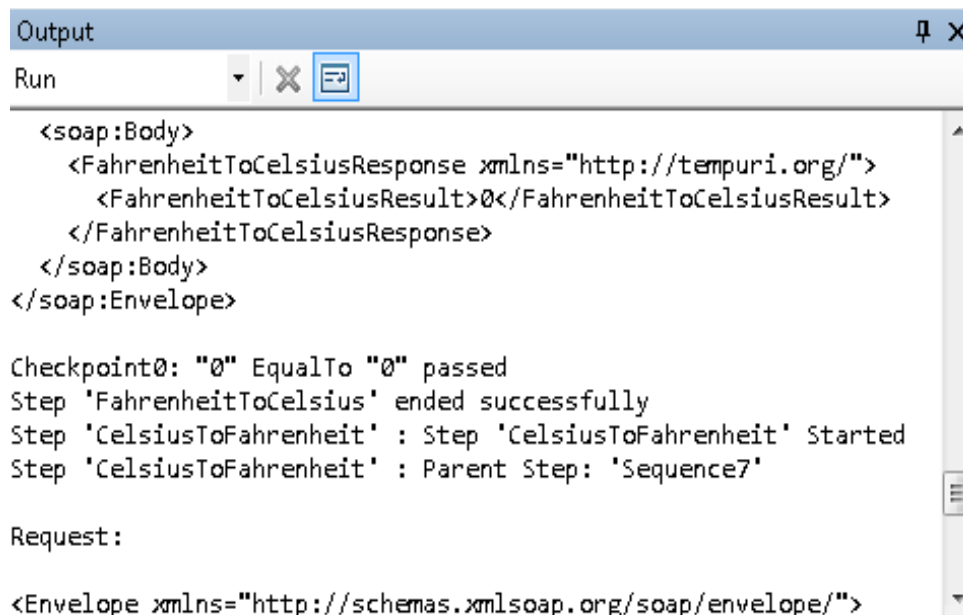
## Property Sheet

The Property Sheet displays a series of tabs showing the properties, schemas, and events for the step selected in the canvas. It provides several views that allow you to set and create step properties and define handler events. It also provides action buttons that allow you to data drive properties and load files.



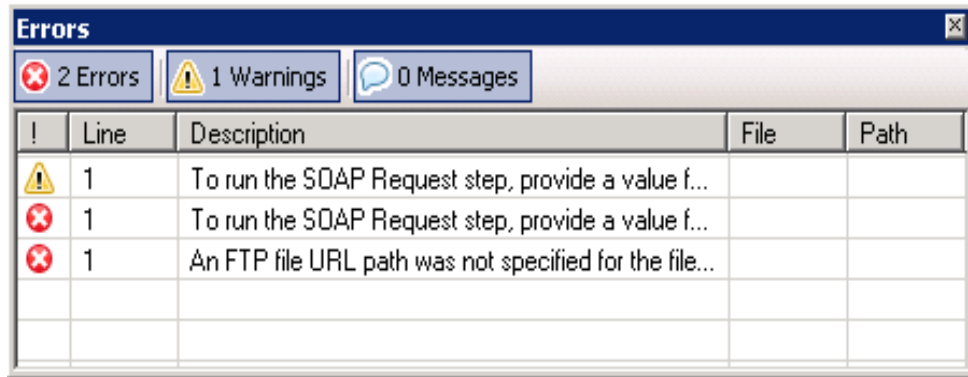
## Output Tab

The Output tab enables you to view the output messages that were generated while compiling and running the test.



## Error Tab

The Error tab enables you to view a list of the errors generated during the test run, and their level of severity: Message, Warning, or Error.



Errors				
2 Errors 1 Warnings 0 Messages				
!	Line	Description	File	Path
!	1	To run the SOAP Request step, provide a value f...		
✖	1	To run the SOAP Request step, provide a value f...		
✖	1	An FTP file URL path was not specified for the file...		

[Back To Index](#)

*To be continued in next issue...*

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](mailto:teatimewithtesters@gmail.com)

\* Conditions Apply



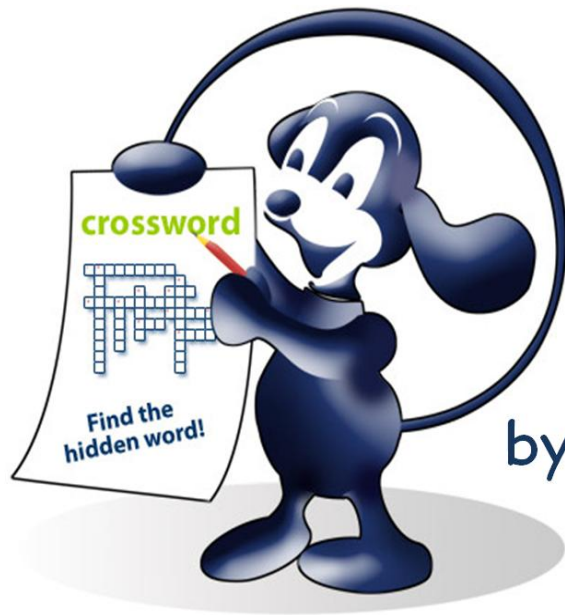
**Karthik Subramanian** is the Senior Technical Solutions Consultant at Hewlett Packard (HP). Prior to that he was with Mercury Interactive; serving as the internal Product Expert on Automation tools.

He holds numerous industry certifications including: Mercury QTP CPC, WR CPS, ISTQB, Sun SCJP and Microsoft MCP. He has also presented at numerous conferences including: HP Meet the Expert Sessions, HP Software Universe and Mercury Software World.

He also holds a masters degree from University of California, Davis and is a graduate of the Indian Institute of Technology (I.I.T).

# Testing PUZZLES

by Sebi



Claim your **Smart Tester of The Month** Award. Send us an answer for the Puzzle and Crossword bellow b4 10<sup>th</sup> Nov 2011 & grab your Title.

Send -> [teatimewithtesters@gmail.com](mailto:teatimewithtesters@gmail.com) with Subject: Testing Puzzle

Gear up guys.....

**It's Time To Tease your Testing Bone**



# Puzzle “Play Around It 😊”

"Find a website (please ask the owner before if its not a problem), where entering a large number of char "/" instead basic url is causing relevant issue to the website(slowing down, malfunction, security etc)

Example:

Using <http://www.google.co.uk//////////i/mghp?hl=en&tab=wi> causes the site to load the images more slowly (this is unproven, used for the purpose of giving an example)



## Biography



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

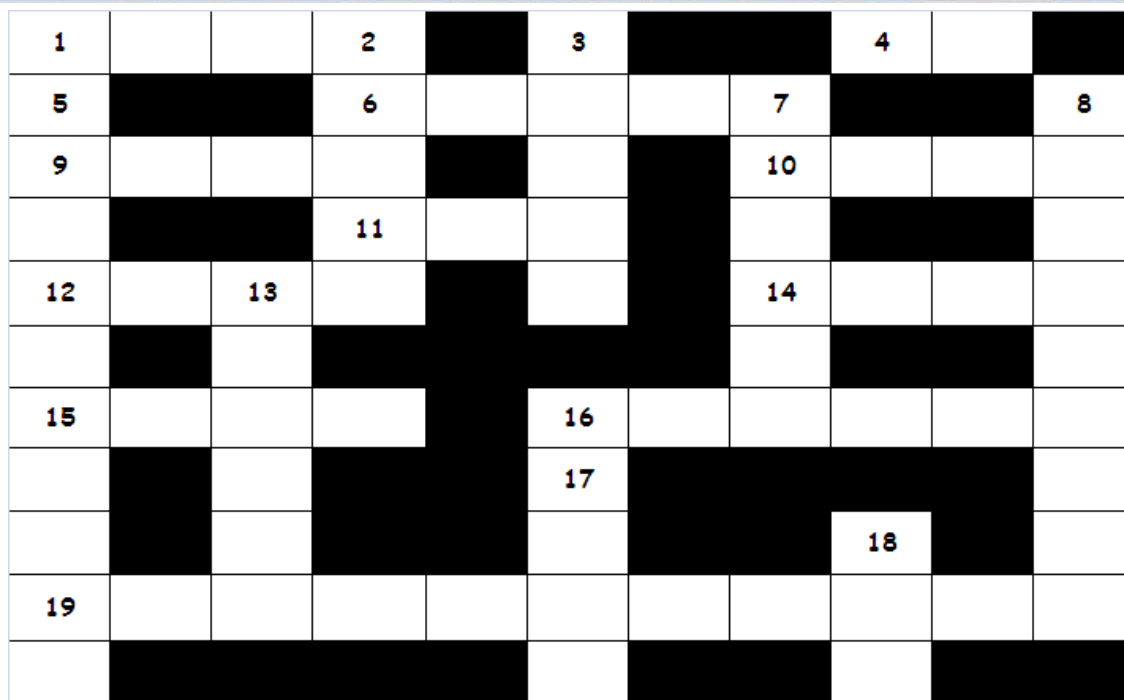
He considers himself a context-driven follower and he is a fan of exploratory testing.

He tweets as @testalways.

You can find some interactive testing puzzles on his website [www.testalways.com](http://www.testalways.com)



# TESTING CROSSWORD



## Horizontal:

1. Most popular Agile coach and practitioner in the world, first name (4)
4. Short form of Quality Assurance (2)
6. To identify and correct mistakes in a computer program (5)
9. The short form of Capability Maturity Model (3)
10. It is the fastest test harness software to learn (3)
11. The first name - It is a software V&V (Testing) company based in Bangalore, India (3)
12. Test Data selection technique, in short form (2)
14. Chief Financial Officer of world's largest marketplace for Software Testing Services, first name (3)
15. A document describing the conduct and results of the testing carried out for a system or system component, the first word (4)
16. CEO and Founder of PushToTest, the first name (5)
19. The first name - Producing tests for the behaviour of an implementation to be sure it provides the portability, interoperability, and/or compatibility a standard defines (11)

## Vertical:

2. Testers tries to break the system by randomly trying the system's functionality, the first name (5)
3. A tool to use web application to help you keep tabs on the time you spend on different projects (5)
5. Testing to verify a product meets customer specified requirements (10)
7. A test automation tool for ajax applications, in short form (5)
8. Checks for memory leaks or other problems that may occur with prolonged execution, the first name (9)
13. It will allow for an abstraction of Test Cases specification and Test Case automation execution (6)
17. It is a tool to provides issue tracking and project tracking for software development teams to improve code quality and the speed of development (4)
18. Short form of Multinational Company (3)

# Answers for Last Month's Crossword:

T	E	S	T	I	T	E	M		W
E		A				A			H
S	O	F	T	W	A	R	E		I
T		S		E			M		T
M				B					E
A		G		L	I	T	M	U	S
K		N		O		C			D
E	T	A		A		W	A		L
R		T		D					C
	C	S	T		B	E	T	A	

Answers for Testing Puzzle:

1. <http://iamtheproudownerofthelongestlongestlongestdomainnameinthisworld.com/>
2. <http://thelongestlistofthelongeststuffatthelongestdomainnameatlonglast.com/>



*We appreciate that you  
"LIKE" US!*



Join us on Facebook.

You are just a CLICK AWAY





**Every Tester**  
**who reads Tea-time with Testers,**  
**Recommends it to friends and**  
**colleagues .**

**What About You ?**

## Our Testimonials

Tea-time with Testers! It is a lovely publication that regularly runs many good articles. I tend to forward links to my team when there are ideas that could benefit them.

- Peter Walen



Thanks very much for sharing the best magazine with me! Looking forward~~~

-Joanna

I am highly impressed with the content and your style guys. You people are doing great job.

Cheers !

- Shobhana Narang

I recently started reading your magazine, and let me say it's amazing. I am planning to write few articles in future.

I really appreciate for the team effort that you guys contributing towards the magazine. Keep it up !

- Jamuna Reddy

Mind blowing stuff. Thanks for creating such a lovely book.

-Sudhir Jagannath

Great Magazine ! Looking forward to learn a lot !

- Suchibrata Acharya

I love your publication and just can't wait to read it each month.

- Rob Irving

Really interested to get your magazine and I will compliment to my level best.

- Mohan raj Rajendran


This is the coolest magazine I have ever read on software testing. Everything about it is just brilliant and nicely put together. Great Job.

-Meenakshi Pandit



# You ask...

# We'll help...!



If you have any questions related to the field of Software Testing, do let us know. We shall try our best to come up with the resolutions.

- Editor

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

We are just a mail away: [teatimewithtesters@gmail.com](mailto:teatimewithtesters@gmail.com)



# in ne>xt issue

articles by -

A photograph of a metal tag with the text "IT'S ALWAYS TEA-TIME" engraved on it. The tag is attached to a chain. Below the tag is a small metal clock with a circular face and a small tag attached to it that says "TEA".

IT'S  
ALWAYS  
TEA-TIME

Jerry Weinberg

T Ashok

Joel Montvelisky

Michael Larsen

Darren McMillan

Anurag Khode

and others

# 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- Etsy

## 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

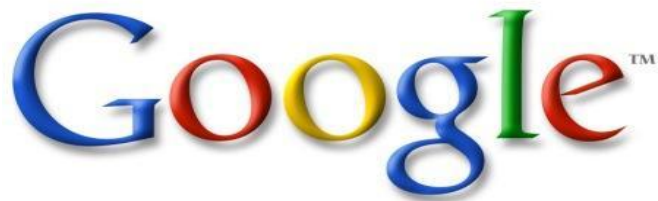


Gautam

*// Karmanye vadhikaraste ma phaleshu kadachina |  
Karmaphalehtur bhurma te sangostvakarmani //*



To get **FREE** copy ,  
Subscribe to our group at



Join our community on



Follow us on



[www.teatimewithtesters.com](http://www.teatimewithtesters.com)

