

THE INTERNATIONAL MONTHLY FOR NEXT GENERATION TESTERS

Tea-time with Testers

APRIL - MAY 2015 | YEAR 5 ISSUE III

Jerry Weinberg

The Eight Fs of Software Failure

Paul Gerrard

Internet of Everything -Test Strategy

Diwakar Menon

The Tester's Kaleidoscope for IoT

Justin Rohrman

Testing the Internet of Things

Jim Holmes

Master the Essentials of UI Test Automation

T Ashok

Beauty is Only Skin-deep

Cullyn Thomson

Bringing Manual and Automated Testing Together

Joel Montvelisky

Peripheral Vision and Peripheral Testing

Sujata Verma

Open Source Tools for Networking Devices Validations



Over a Cup of Tea with Rajesh Mathur



TeamQualityPro

SEE EVERYTHING



Defects / Testing

- Real time reporting
- Instantaneous data gathering
- Objective, vetted data
- Comprehensive data
- Data that mirrors your process
- Data that is always current

- Manual reporting ○
- Data warehouse projects ○
- Subjective data ○
- Missing data ○
- Not aligned data ○
- Wrong time frame data ○



testomaton

Quality Assurance via Automation

Software testing startup specializing in test automation services, consulting & training for QA teams on how to build and evolve automation testing suites.

SERVICES

Test System Analysis and work estimation

Review of client's system (either in development or on early stage) to produce a Test Plan document with needed test cases and scenarios for automation testing.

Tests creation and Automation

Development of test cases (in a test plan) and Test Scripts to execute the test cases.

Regression and Test evolution

Development of Regression test suites and New Features suites, including test plans and test scripts or maintenance of existing.

Architecture Consulting

Review of software architecture, components and integration with other systems (if applicable).

Trainings

Depending on the particular need, we can provide training services for: Quality Assurance theory and best practices, Java, Spring, Continuous Integration, Groovy, Selenium and frameworks for QA. For further information please check our web site.



We enjoy putting our experience to your service. Our know-how allows us to provide consultation services for projects at any stage, from small up to super-large. Due to our range of expertise we can assist in software architecture and COTS selection; review of software designs; creation of testing suites and test plans with focus on automation; help building quality assurance teams via our extensive training curriculum.

look us up: www.testomaton.com - twitter: @testomaton



TEA-TIME WITH TESTERS

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

Created and Published by:

Tea-time with Testers.

B2-101, Atlanta, Wakad Road

Pune-411057

Maharashtra, India.

Editorial and Advertising Enquiries:

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

Lalit: (+91) 8275562299

Pratik: (+49)15215673149

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 or claims made by advertisers in this ezine do not necessarily reflect those of the editors of **Tea-time with Testers**.

Editorial



So, are they killing testing (again)?

While DevOps is emerging as new favorite of the fellas in Software Development, it seems that some of my tester friends have already lost hope for their future. Yes, I'm talking about those testers who don't know coding or don't like to code.

Whether or not DevOps really means the end of 'testing era' is another topic of discussion, I have experienced it to some extent that some programmers (or Dev managers for that matter) with typical anti-testing mindset have already started celebrating it like "We don't need, no testers" thingy.

Being practitioner of craft of software testing, I'm personally not much worried about such threats to existence of testing (which keep on coming every few years with new and newer names) for I have realized that it's a pipe dream of those people who think it is possible to develop software without meaningful testing. Well, it's not entirely their fault. As long as people keep on confusing [checking with testing](#), they are likely to keep on dreaming about it. Funny thing is, organizations try new and newer things, celebrate the adoptions to new things in market but nobody really talks about failures when those attempts fail.

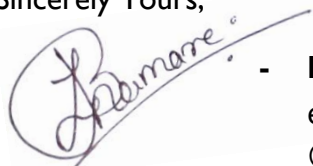
Well, I'm not much bothered about that either. So, why this topic? I'm writing this for those skilled testers who seem to be afraid of their future or are trying to switch to programming just to survive. I admit that programming is a nice to have skill for testers, I don't personally think it's a good idea to convert good tester into bad programmer or good programmer into bad tester. Pradeep Soundararajan had once rightly said that testing is a team sport and you need to have testers with all sort of skills in your team. Of course technical skills, automation skills, programming skills have advantages of their own if testers know it but current over emphasis on these skills in market is what concerning me. Why? Because this thing called "Internet of Things" is standing right next to us and in my opinion it warrants some highly skilled thinking testers to be in your team who will dedicatedly do the "thinking" part to catch what lies beyond the code.

In this issue we are publishing some interesting insights around Internet of Things and I hope that it will help our tester friends to figure out what future holds for them. Special thanks to Paul Gerrard, Diwakar Menon and Justin Rohrman for their articles on IoT in this issue. As far as doing automation for checking part of testing is concerned, Jim Holmes has been writing wonderful articles on this topic. Don't forget to check out his series from part I. For those who are already aware of test automation, Georgios's article series would be a bonus. And don't forget again to check what Cullynn Thomson has to say around doing manual and automated testing together.

While the right balance of automated checks and skilled exploratory testing is what I personally think is the golden way, one thing looks clear to me that with Internet of Things coming in, the epicenter of SDLC is going to be a human tester with excellent testing skills. If you are one of those afraid testers, don't declare your horses dead yet! But you must not let your tools get worn out either.

On side note, there have been demands for reducing the content in our monthly issues. On other hand, there are readers who can't wait for each issue to come. A temporary solution we have found to meet both the demands is to club the issues (like this one) when needed. We are working on some cool idea and hopefully we will be able to find/implement permanent solution to this problem in next few months. Until then...

Sincerely Yours,



- **Lalitkumar Bhamare**

editor@teatimewithtesters.com

@Lalitbhamare / @TtimewidTesters



QuickLook



Editorial

What's making News?

Tea & Testing with Jerry Weinberg

Speaking Tester's Mind

Testing the Internet of Things -21

Making the User Story Trust Fall - 23

Internet of Everything -Test Strategy -26

The Tester's Kaleidoscope for IoT - 40

In the School of Testing

Open Source Tools for Networking Devices
Validations – 46

Road to Test Automation – 51

Brining Manual and Automated Testing Together
– 56

Master the Essentials of UI Test Automation - 59

Peripheral vision and peripheral testing – 72

T' Talks

Beauty is only skin-deep – 70

Over a Cup of Tea with Rajesh Mathur

Family de Tea-time with Testers





What's making News?

New Approach to Automated API Security Testing Simplifies Digital Safety for Modern Web Service Teams

New Secure Pro, part of SmartBear's Ready! API Platform, Democratizes Security Testing of Back-End APIs, Raising Level of Excellence in Safety for Software Teams

SOMERVILLE, Mass. – SmartBear Software, the leader in software quality tools for the connected world, today announced Ready! API Secure Pro, a new approach for API security testing that simplifies digital safety for modern web service teams. Secure Pro, part of SmartBear's Ready! API family, is a brand new approach to API security testing, covering common areas of risk for both modern REST and traditional SOAP Web services.

With the broadening adoption of APIs, security over Web services is paramount. As data breaches in back-end systems become increasingly prevalent, deep security testing remains a luxury item for many software teams. API providers often do not have the combination of security analysis skills and technical context on their APIs within the same individuals. This skill deficit represents a significant risk in APIs that can only be satisfied with conscious security testing by those who know their APIs best.

SmartBear's new Secure Pro empowers development and testing teams with an affordable way to quickly scan APIs for security problems as part of their existing delivery lifecycle. With this new ability to perform security testing earlier and more frequently, software teams can minimize risk of data breaches in production and be prepared for successful compliance audits as part of their normal operations.

While APIs simplify how we connect everything, they also represent the largest increase in surface area for potential hackers since the dot com boom. Teams who build and ship APIs as part of great digital

experiences need to rethink traditional approaches to security. "Simply embedding a security expert into the design organization will overwhelm the healthy tension and likely make it even more difficult to get sufficiently secure products out the door," writes Tyler Shields of Forrester Research. "Instead, both design and development teams must operate with security as a fundamental base assumption."**

With Secure Pro, SmartBear provides development and testing teams with the power to:

- Run threat analysis over REST and SOAP Web services to identify common security problems
- Analyze patterns of API communication and review suggestions on corrective actions
- Scan Web services for non-API security considerations such as sensitive file exposure
- Run security scans over either a single request or multi-step transactions
- Ensure that APIs work properly, are safe and achieve high performance and scalability goals

In addition to new Secure Pro, this release of the Ready! API platform also includes support for distributed testing and server monitoring in LoadUI NG, as well as integration to SmartBear's application quality and performance monitoring service, AlertSite UXM.

For more information on Secure Pro, visit: <http://smartbear.com/product/ready-api/secure/overview>

To see more about SmartBear's API readiness platform, Ready! API 1.3, visit:

<http://smartbear.com/product/ready-api/overview/>

**Cited from Create And Support The New Security Designer Role, April 2015, Forrester Research

(Access requires subscription)

About SmartBear Software

As the leader in software quality tools for the connected world, SmartBear supports more than three million software professionals and over 25,000 organizations in 194 countries that use its products to build and deliver the world's greatest applications. With today's applications deploying on mobile, Web, desktop, Internet of Things (IoT) or even embedded computing platforms, the connected nature of these applications through public and private APIs presents a unique set of challenges for developers, testers and operations teams. SmartBear's software quality tools assist with code review, functional and load testing, API readiness as well as performance monitoring of these modern applications. For more information, visit: <http://www.smartbear.com>

BroadPR, Inc

I got introduced to Rajesh Mathur through his work via Association for Software Testing (AST). Rajesh has also written some interesting articles for Tea-time with Testers in past.

He is an International speaker, blogger and writer with nearly 20 years of IT experience in Management Consulting, Engagement Management, Test Management and Testing Practice Development. Rajesh is a senior member of Australian Computer Society and was awarded with the Fellowship of the Hong Kong Computer Society for his contribution to the International testing community and for promoting software testing in Hong Kong's IT industry.

For among many other things I have special respect for Rajesh for his contribution towards community and for the kind of admiration he has for our work around Tea-time with Testers. I got to discuss some important aspects around testing, test management and leadership with Rajesh and what you see below is the result of it. I'm sure that you'll enjoy this interview like many others we have published in past.

Enjoy!

- Lalitkumar Bhamare

Over a Cup of Tea with Rajesh Mathur



It's our pleasure talking with you today, Rajesh. Please tell our readers about your journey in software testing.

It is my pleasure indeed. My journey in software testing has been like an adventurer or explorer who keeps on exploring and learning new things. I have worked with start-ups as well as blue-chip companies and so far I have enjoyed wherever I have worked. There is always something new that you can learn and enjoy.

I started my career as a developer and I guess I was not a great developer. In the actual fact when I started working, testing was not as prominent as it is today. No one was seen as a career tester and the craft was considered a low level activity. Personally, I knew that I was better at looking at things from a different angle and that helped me transitioning into testing. When I discovered what real testing is, I just fell completely for it.

You have vast experience of testing in multiple domains. How much do you think domain knowledge is important to be a good tester?

It is a very broad question and the answer I think is, "It depends". Good testing does not depend on the knowledge of a particular domain, but very good testing probably does. You can compare domain specialization to Medical profession. One can be either a General Practitioner or can consider getting a specialization by studying more. General practice will probably ensure that you will have a job in any economic situation. However, specialization will probably offer you better remuneration as you have command over a specific area. If market situation is not good, a specialist may not be able to secure the work he or she is seeking.

Another aspect of this is, if you do not know about a specific area, you cannot test it. For example, if you are asked to run Avionics tests, you will need to know what avionics is otherwise you will not be able to test it. However, what is more important is to learn quickly. So even if you are not a domain specialist, if you are able to learn quickly, you will be able to add value. In my case, I try to read, study, and learn as much as I can before testing for a particular domain. My reading, questioning and note taking habits help me in this.

Out of different roles you have worked on in last 20 years, which one did you enjoy working most and why?

It's a difficult one. I guess I liked most roles I worked on. If I don't like something in a role, I try to find something that makes it interesting. Each role brings something interesting or challenging. One role that I enjoyed most was at Cathay Pacific Airways where I worked on e-Enabled Aircraft programme. This project was an aircraft avionics project intended to implement real time connectivity between the ground and aircraft among other objectives like reducing the weight and making the aircraft paperless. In order to create a strategy for testing the project, I had to study & learn a lot about aircraft engineering, avionics and a lot more about regulations in the aviation industry. I got the opportunity to fly inside the flight deck (cockpit), understand what aircraft actually look like under the chassis and how complex these beautiful machines are. I loved airplanes since my childhood and this role gave me an opportunity to learn about them from up close.

We are curious to know about your contribution in Hong Kong's testing industry for which you were rewarded with fellowship.

Yeah, it was an absolute pleasure to be awarded with the Fellowship of the Hong Kong Computer Society (HKCS). When I moved to Hong Kong from the UK in 2010, I started looking for people who were interested in testing. I soon realized that the Hong Kong testing industry was still not very mature and in the actual fact it was very much inclined towards the theoretical aspects of testing rather than cognitive aspects. The first testing meetup I arranged got hardly any response from testers. I had to ask few team members to come over for free drinks & food while we discussed about testing. In all, there were seven people and only one came to discuss testing, rest were motivated by free drinks.

... CONTINUED ON [PAGE 64](#)

Tea & Testing



with

Jerry Weinberg

The Eight Fs of Software Failure

It doesn't have to be that way

Disaster stories always make good news, but as observations, they distort reality. If we consider only software engineering disasters, we omit all those organizations that are managing effectively. But good management is so boring! Nothing ever happens worth putting in the paper. Or almost nothing. Fortunately, we occasionally get a heart-warming story such as Financial World telling about Charles T. Fisher III of NBD Corporation, one of their award-winning CEO's for the Eighties:

"When Comerica's computers began spewing out erroneous statements to its customers, Fisher introduced Guaranteed Performance Checking, promising \$10 for any error in an NBD customer's monthly statement. Within two months, NBD claimed 15,000 new customers and more than \$32 million in new accounts."

What the story doesn't tell is what happened inside the Information Systems department when they realized that their CEO, Charles T. Fisher III, had put a value on their work. I wasn't present, but I could guess the effect of knowing each prevented failure was worth \$10 cash.

The Second Rule of Failure Prevention

One moral of the NBD story is that those other organizations do not know how to assign meaning to their losses, even when they finally observed them. It's as if they went to school, paid a large tuition, and failed to learn the one important lesson—the First Principle of Financial Management, which is also the Second Rule of Failure Prevention:

A loss of X dollars is always the responsibility of an executive whose financial responsibility exceeds X dollars.

Will these other firms ever realize that exposure to a potential billion dollar loss has to be the responsibility of their highest ranking officer? A programmer who is not even authorized to make a long distance phone call can never be responsible for a loss of a billion dollars. Because of the potential for billion dollar losses, reliable performance of the firm's information systems is a CEO level responsibility.

Of course I don't expect Charles T. Fisher III or any other CEO to touch even one digit of a COBOL program. But I do expect that when the CEOs realize the value of trouble-free operation, they'll take the right CEO-action. Once this happens, this message will then trickle down to the levels that can do something about it—along with the resources to do something about it.

Learning from others

Another moral of all these stories is that by the time you observe failures, it's much later than you think. Hopefully, your CEO will read about your exposure in these case studies, not in a disaster report from your office. Better to find ways of preventing failures before they get out of the office.

Here's a question to test your software engineering knowledge:

What is the earliest, cheapest, easiest, and most practical way to detect failures?

And here's the answer that you may not have been expecting:

The earliest, cheapest, easiest, and most practical way to detect failures is in the other guy's organization.

Over more than a half-century in the information systems business, there have been many unsolved mysteries. For instance, why don't we do what we know how to do? Or, why don't we learn from our mistakes? But the one mystery that beats all the others is why don't we learn from the mistakes of others?

Cases such as those cited above are in the news every week, with strong impact on the general public's attitudes about computers. But they seem to have no impact at all on the attitudes of software engineering professionals. Is it because they are such enormous losses that the only safe psychological reaction is, "It can't happen here (because if it did, I would lose my job, and I can't afford to lose my job, therefore I won't think about it)."

The Significance of Failure Sources

If we're to prevent failures, then we must observe the conditions that generate them. In searching out conditions that breed failures, I find it useful to consider that failures may come from the following eight F's: frailty, folly, fatuousness, fun, fraud, fanaticism, failure, and fate. The following is a brief discussion of each source of failure, along with ways of interpreting its significance when observed.

But before getting into this subject, a warning. You can read these sources of failure as passing judgment on human beings, or you can read them as simply describing human beings. For instance, when a perfectionist says "people aren't perfect," that's a condemnation, with the hidden implication that "people should be perfect." Frankly, I don't think I'd enjoy being around a perfect person, though I don't know, because I've never met one. So, when I say, "people aren't perfect," I really mean two things:

"People aren't perfect, which is a great relief to me, because I'm not perfect."

"People aren't perfect, which can be rather annoying when I'm trying to build information system. But it will be even more annoying if I build my information system without taking this wonderful imperfection into account."

It may help you, when reading the following sections, to do what I did when writing them. For each source, ask yourself, "When have I done the same stupid thing?" I was able to find many examples of times when I made mistakes, made foolish blunders, made fatuous boo boos, had fun playing with a system and caused it to fail, did something fraudulent (though not, I hope, illegal or immoral), acted with fanaticism, or blamed fate for my problems. Once, I actually even experienced a hardware failure when I hadn't backed up my data. If you haven't done these things yourself (or can't remember or admit doing them), I'd suggest that you stay out of the business of managing other people until you've been around the real world a bit longer.

Frailty

Frailty means that people aren't perfect. They can't do what the design calls for, whether it's the program design or the process design. Frailty is the ultimate source of software failure. The Second Law of Thermodynamics says nothing can be perfect. Therefore, the observation that someone has made a mistake is no observation at all. It was already predicted by the physicists.

It was also measured by the psychologists. Recall case history 5, the buying club statement with the incorrect telephone number. When copying a phone number, the programmer got one digit incorrect. Simple psychological studies demonstrate that when people copy 10-digit numbers, they invariably make mistakes. But everybody knows this. Haven't you ever copied a phone number incorrectly?

The direct observation of a mistake has no significance, but the meta-observation of how people prepare for mistakes does. It's a management job to design procedures for updating code, acknowledging facts of nature, and seeing that the procedures are carried out. The significant observation in this case, then, is that the managers of the mail-order company failed to establish or enforce such procedures.

In Pattern 1 and Pattern 2 organizations, for instance, most of the hullabaloo in failure prevention is directed at imploring or threatening people not to make mistakes. This is equivalent to trying to build a kind of perpetual motion machine—which is impossible. Trying to do what you know is impossible is fatuousness, which we will discuss in a moment.

After a mistake happens, the meta-observation of the way people respond to it can also be highly significant. In Pattern 1 and Pattern 2 organizations, most of the response is devoted to establish blame and then punishing the identified "culprit." This reaction has several disadvantages:

- It creates an environment in which people hide mistakes, rather than airing them out.
- It wastes energy searching for culprits that could be put to better use.
- It distracts attention from management responsibility for procedures that catch failures early and prevent dire consequences.

The third point, of course, is the reason many managers favor this way of dealing with failure. As the Chinese sage said,

When you point a finger at somebody, notice where the other three fingers are pointing.

Folly

Frailty is failing to do what you intended to do. Folly is doing what you intended, but intending the wrong thing. People not only make mistakes, they also do dumb things. For example, it's not a mistake to hard code numerical billing constants into a program as was done in the public utility billing cases. The programs may indeed work perfectly. It's not a mistake, but it is ignorant because it may cause mistakes later on.

Folly is based on ignorance, not stupidity. Folly is correctable, whereas frailty is not. For instance, it is folly to pretend not be frail, that is, to be perfect. Either theoretical physics or experience in the world can teach you that nobody is perfect.

In the same way, program design courses can teach you not to hard code numerical constants. Or, you can learn this practice as an apprentice to a mentor, or from participating in code reviews where you can observe good coding practices. But it's management's job to establish and support training, mentoring, and technical review programs. If these aren't done, or aren't done effectively, then you have a significant meta-observation about the management of failure.

Fatuousness

It is worse than folly to manage a foolish person and not provide the training or experience needed to eradicate the foolishness. We call such behavior "fatuousness." ("Utter stupidity" would be better, but it doesn't start with F.) Fatuousness is utter stupidity, or being incapable of learning. Fatuous people—which occasionally includes each of us—actively do stupid things and continue to do them, time after time. For example,

Ralston, a programmer, figures out how to bypass the configuration control system and zaps the "platinum" version of an about-to-be-released system. The zap corrects the situation he was working on, but results in a side-effect that costs the company several hundred thousand dollars.

The loophole in configuration control is fixed, but on the next release, Ralston figures out a new way to beat it. He zaps the platinum code again, producing another 6-figure side effect.

Once again, the new loophole is fixed. Then, on the third release, Ralston beats it again, although this time the cost is only \$45,000.

The moral of this story is clear. The fatuous person will work very hard to beat any "idiot-proof" system. Indeed, there is no such thing as an "idiot-proof" system, because some of the idiots out there are unbelievably intelligent.

There's no protection against fatuous people in a software engineering organization except to move them into some other profession. What significance do you make of this typical situation?

Suppose you were Ralston's manager's manager. Hunt, his immediate manager, complains to you, "This wouldn't have happened if Ralston hadn't covertly bypassed our configuration control system. I don't know what to do about Ralston. He goes out of his way to do the wrong thing, beating all our systems of protection. And he's done this three times before, at least."

What was the significant part of this story? Ralston, of course, has to be moved out, but that's only the second most important part. Hunt—who has identified a fatuous employee and hasn't done anything about it—is doubly fatuous. Hunt needs to be recycled out of management into some profession where his utter stupidity doesn't carry such risk. If you delay in removing Hunt until he's done this with three employees, what does that make you?

Fun

Ralston's story also brings up the subject of fun. Some readers will rise to the defense of poor Ralston, saying, "He was only trying to have a little fun by beating the configuration control system." Well, I'm certainly not against fun, and if Ralston wants to have fun, he's entitled to it. But the question Ralston's manager has to ask is, "What business are we in?" If you're in the business of entertaining your employees at the cost of millions, then Ralston should stay. Otherwise, he'll have to have his fun hacking somewhere else.

In the actual situation, Ralston wasn't trying to have fun—at least that wasn't his primary motivation. He was, in fact, trying to be helpful by putting in a last minute "fix." Well-intentioned, but fatuous, people like Ralston are not as dangerous as people who are just trying to have a good time. Hunt could have predicted what Ralston was going to be helpful, but

Nobody can predict what somebody else will consider "fun."

Here are some items from my collection of "fun" things that people have done, each of which has resulted in costs greater than their annual salary:

- Created a subroutine that flashed all the lights on the mainframe console for 20 seconds, then shut down the entire operating system.
- Created a virus that displayed a screen with Alfred E. Neumann saying "What, me worry?" in every program that was infected.

- Altered the pointing finger in a Macintosh application to point with the second finger, rather than the index finger. The testers didn't notice this obscene gesture, but thousands of customers did.
- Diddled the print spooler so that in December, "Merry Christmas" was printed across a few tens of thousands of customer bills, as well as all other reports. The sentiment was nice, but happened to obliterate the amount due, so that customers had to call individually to find out how much to pay.

The list is endless and unpredictable, which is why fun is the most dangerous of all sources of failure. There are only two preventives: open, visible systems and work that is sufficient fun in and of itself. That's why fun is primarily a problem of Pattern 2 organizations, which seldom meet either of those conditions.

Fraud

Although fun costs more, software engineering managers are far more afraid of fraud. Fraud occurs when someone illegally extracts personal gain from a system. Although I don't mean to minimize fraud as a source of failure, it's an easier problem to solve than either fun or fatuousness. That's because it's clear what kind of thing people are after. There are an infinite number of ways to have fun with a system, but only a few things worth stealing.

I suggest that any software engineering manager be well read on the subject of information systems fraud, and take all reasonable precautions to prevent it. The subject has been well covered in other places, so I will not cover it further.

I will confess, however, to a little fraud of my own. I have often used the (very real but minimal) threat of fraud to motivate managers to introduce systematic technical reviews. I generally do this after failing to motivate them using the (very real and significant) threat of failure, folly, fatuousness, or fun.

Fanaticism

Very infrequently, people try to destroy or disrupt a system, but not for direct gain. Sometimes they are seeking revenge against the company, the industry, or the country for real or imagined wrongs done to them. Fanaticism like this is very hard to stop, if the fanatic is determined, especially because, like "fun," you never know what someone will think is an offense that requires revenge.

Fanaticism, like fraud, is a way of getting the attention of management. With reasonable precautions, however, the threat of terrorism can be reduced far below that of frailty. Frailty, however, lacks drama. In any case, many of the actions that protect you against frailty will also reduce the impact of terrorism. Besides, I cannot offer you any useful advice on how to observe potential terrorists in your organization. That would be "profiling."

Failure (of Hardware)

When the hardware in a system doesn't do what it's designed to do, failures may result. To a great extent, these can be overcome by software, but that is beyond the scope of this book. Fifty years ago, when programmers complained about hardware failures, they had a 50/50 chance of being right. Not today. So, if you hear people blaming hardware failures for their problems, this is significant information. What it signifies can be chosen from this list, for starters:

1. There really aren't significant hardware failures, but your programmers need an alibi. Where there's an alibi, start looking for what it's trying to conceal.
2. There really are hardware failures, but they are within the normally expected range. Your programmers, however, may not be taking the proper precautions, such as backing up their source code and test scripts.
3. There really are hardware failures, and you are not doing a good job managing your relationship with your hardware supplier.
4. Failure attributed to hardware may actually be caused by human error—unexpected actions on the part of the user. These are really system failures.

Fate

This is what most bad managers think is happening to them. It isn't. When you hear a manager talking about "bad luck," substitute the word "manager" for "luck." As they say in the Army, "There are no bad soldiers, only bad officers."

Biography

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



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

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

To know more about Gerald and his work, please visit his Official Website [here](#).

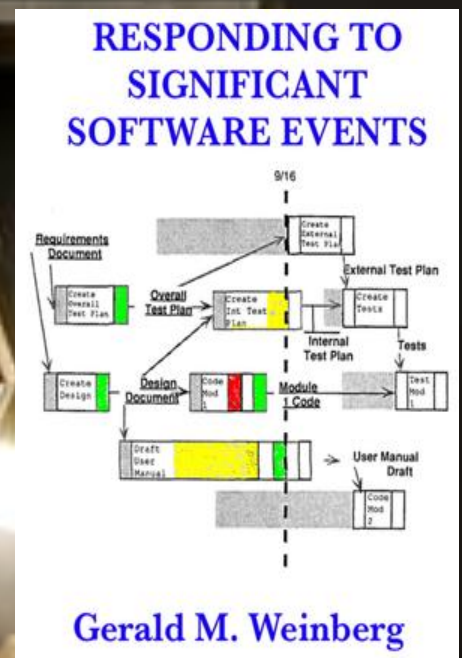
Gerald can be reached at hardpretzel@earthlink.net or on twitter @JerryWeinberg

Reviewers say "This book focuses on an issue of huge importance to software managers: how to respond appropriately to people (clients, bosses, team members) in difficult, emotionally charged situations."

Reviewer Keith Collyer wrote that he "didn't see how anyone can consider themselves interested in software quality without having some of Gerald Weinberg's books on their shelves (preferably well-thumbed)."

We strongly recommend you to read **Responding to Significant Software Events**. Its sample can be **read online**.

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



TTWT Rating: ★★★★★

The Bundle of Bliss

Buy Jerry Weinberg's all testing related books in one bundle and at unbelievable price!

The Tester's Library

Sold separately, these books have a minimum price of \$83.92 and a suggested price of \$83.92...



The suggested bundle price is **\$49.99**, and the minimum bundle price is...

\$49.99!

Buy the bundle now!

The Tester's Library consists of eight five-star books that every software tester should read and re-read. As bound books, this collection would cost over \$200. Even as e-books, their price would exceed \$80, but in this bundle, their cost is only \$49.99.

The 8 books are as follows:

- Perfect Software
- Are Your Lights On?
- Handbook of Technical Reviews (4th ed.)
- An Introduction to General Systems Thinking
- What Did You Say? The Art of Giving and Receiving Feedback
- More Secrets of Consulting
- Becoming a Technical Leader
- The Aremac Project

[Know more about this bundle](#)

TEA-TIME WITH SMARTBEAR



About this column...



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

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

Testing the Internet of Things

- by **Justin Rohrman**

Microsoft today unveiled their latest operating system, Windows 10. We quickly put a list of top features that could be essential to testing applications operating on this new version.

One Platform Across All Devices

The Windows 10 operating system works across multiple platforms i.e. Windows, Windows Phone, and Xbox. Test Cases that are easier to reuse could really help reduce time while testing across these multiple devices.

Project Spartan

Microsoft announced new web browser for Windows 10 "Project Spartan". The browser has all-new rendering engine, which essentially means it will be critical to test existing websites against the new browser. Just like Office, the browser allows for note taking and adding annotations. Additionally, the reading mode in Project Spartan removes the clutter from the page and makes the experience like reading an eBook. Users can even save the page for reading in offline mode. In order to complement what quite possibly could become a popular browser, websites containing blogs, articles, or newsletters definitely need to support this new feature and ensure they are able to sync the saved reading list across multiple devices (PC and phone) that are using the new browser.

Continuum Mode

Windows 10 allows users to switch between a regular desktop and a notebook, announced today as "Continuum Mode". Users just need to detach keyboard from the desktop and then use gestures to interact with the touchscreen as if it were a notebook. Apps, which are built for the Windows 10, need to therefore ensure that user interface remains consistent when a switch is made between these modes (notebook-tablet). Additionally, apps built for Windows 10 need to handle, not just mouse controls, but touchscreen movements effectively. Automated testing solutions therefore need to be able to record repeatable and consistent gestures and replay these gestures on multiple devices in the exact same manner every time.

The Cortana API

Microsoft's virtual assistant Cortana API is now integrated with PC. This means the end user can now more effectively leverage voice commands to interact with desktop applications out of the box. During the demo, Joe Belfiore, Corporate VP at Microsoft, demonstrated how Cortana could be used to draft emails, check flights, or even track the weather. With the addition to Cortana, from Windows Phone to Windows PC, desktop applications need to be tested for interactive voice commands. Cortana will also integrate with Project Spartan, allowing users to search and use voice to get quick results on your desktop web applications. Along with Cortana, Microsoft is also releasing a huge number of other APIs such as DirectX

12 graphics API. Testing APIs for correct response and behavior is therefore going to be essential for applications working on Windows 10 operating system.

We commend Microsoft's Windows 10 as a meaningful attempt to minimize fragmentation in the mobile and PC device market, while also bringing further improvements to accessibility. However, testers need to be better prepared to reduce additional effort that might be required to test new and old applications on Windows 10.



Justin has been a professional software tester in various capacities since 2005. In his current role, Justin is a consulting software tester and writer working with Excelon Development. Outside of work, Justin is currently serving on the Association For Software Testing Board of Directors as VP of Education helping to facilitate and develop projects like BBST and WHOSE. He is also a student in the Miagi-Do school of software testing, and facilitate sessions for Weekend Testing Americas.

Justin is deeply interested in software testing and delivery, and also in helping organizations fix problems in measurement and metrics programs.

Tea, Testing and



Making the User Story Trust Fall

Agilefall alert!

I think one of the major misconceptions of moving to TDD, ATDD, or BDD is that the majority of your transition relies on transforming your existing assets from the old format to the new one. The real effort, and value of moving to these processes is in *garnering early feedback from a variety of team members when determining the user stories and acceptance criteria related to them.*

Granted, there are some formatting changes that need to occur to get from a BRD to a User Story. A typical BRD may look like *this*, but a user story is typically a much different structure, such as:

As a user,

I want to be able to save my credit card details securely,

So that I can complete my return check outs easily without security risk.

Of course, there would be *acceptance criteria* associated with this user story as well that would define the level of "done-ness" for this user story.

However, the answer to getting to a user story driven process is not asking the question "which items of the BRD map to the user story". This is because *while the data in a BRD may appear similar to the data that is contained in a user story, the 1. players, 2. context, and 3. collaboration involved in user story creation are usually different from that of BRD creation.*

1. Players

Many times, Business Requirements are authored by internal Subject Matter Experts (SME's) for the functional area being scoped and approved by the development leads who are tasked with building out the components in that particular area. In the user story driven process, a greater variety of contributors are involved up front, with the goal of getting the most input and agreement from the start, rather than focusing on just the people signing off on the feature.

2. Context

Business Requirements creation typically involves SME's/end users and developers negotiating to reach resolution over the functionality that will be delivered in a given release - an agreement of "if you give me this capability, I will be happy." User story creation is often more imaginative, taking the experiences and feedback from a wealth of users to consider not only the ways the system should be used, but the ways it could be used as well. The intent is to innovate and think of unique scenarios that the team can then effort estimate and priority rank to make sure the most "bang for the buck" is achieved in each sprint.

3. Collaboration

Business Requirements creation too often involves only the most knowledgeable SME's and developers, with the reason being that requirements are often being fleshed out to drive months/years of development effort, and the goal is to reach agreement quickly. When the release is finally developed and is being tested, others are unaware of the decisions and compromises that were made, resulting in a large number of change requests. User story creation aims to use an Agile sprint structure to look at a smaller window of development effort, and get more players involved in the process. The combination of the increased collaboration up front and the smaller time window from user story inception to "done-ness" means that risk of a major concern appearing during the test cycles is usually much lower.

Overall, it is my belief that most organizations cannot simply convert their BRD's into user stories and successfully implement an Agile process. While these artifacts may appear to be a user story, they were not developed in the proper fashion. In an Agile process, the group learning and knowledge sharing that is created during the user story creation process is just as important as the documentation itself, so converting a BRD to a user story format will circumvent that process and its value. So, if you are looking to make a transition from waterfall to agile - leave your old documentation behind and commit to making the user story trust fall!



Kevin Dunne is the Director of Product Strategy at QASymphony where he collaborates with customers and industry experts to continue to provide innovative testing tools for agile teams.

You can find out more at www.QASymphony.com

A green, teardrop-shaped object, possibly a pendulum bob, hangs from a thin string. It is positioned over a light-colored sand surface. In the sand, there are intricate, swirling patterns that resemble a sand mandala or a complex drawing. The overall scene is framed by a dark blue border.

Speaking Tester's Mind

- straight from the author's desk

Internet of Everything

– Test Strategy



Introduction

In the first two articles of this series, I set the scene of the Internet of the looked at the evolving architecture of the IoE and speculated on the emerging risks of it [1, 2]. These articles describe the scale of the challenge and introduce a seven-layer architecture that might help you to understand the features that must exist to make the IoE happen. The second article also sets out at a high level some of the risks that we must address.

This is the third instalment – on IoE Test Strategy. At the outset, I have to tell you that figuring out how we test what I will now suggest will be every Internet-connected system component that exists now and forever has proved to be quite difficult! In this article, I won't solve all of the testing challenges involved, but what I can perhaps do is suggest some of the dimensions, some of the influences, and some of the opportunities of the IoE testing problem.

I can only paint a picture of what I believe we will have to test for the next ten to fifteen years and will suggest what testers need to think about and much more tentatively, how they might think – because the challenge we face is quite different from what we have tested before.

Some of the statements I make in this paper derive from a recent trip to Barcelona. I was a visitor to the Mobile World Congress [3] and learned a lot from the people I met there. I was one of 93,000 visitors, apparently.

Scope of the Internet of Everything

When I embarked on this article series, I thought, like many people still do, that the IoT was mostly devices at the 'edge of the internet', but over the course of the last twelve months or so, the scope of what is increasingly called the Internet of Everything has broadened. Some parts of the IT landscape have been brought under the IoE umbrella and quite a few more will be included in time.

The IoE includes the vast number of static devices, mostly sensors, but also mobile devices including cars, trucks, buses, trains, planes, ships and even satellites. It will come to include both every-day and obscure objects that we encounter at home, in cities, our work environment, hospitals and entertainment venues. Naturally, it includes wearable and even human-embedded devices, mobile phones and tablet computers too. And, after all that is taken into account, much of the data that is captured by these devices will be stored, processed and analysed by cloud-based servers that integrate with other cloud-based services, partners, payment systems and other popular services and the legacy systems of large companies.

I expect that the Internet of Everything really will become an apt description. In fact, in time, the Internet will become shorthand for the IoE.

With this seeming inevitability in mind, the test strategy for the IoE really does become the test strategy for everything on the internet.

The Biggest Thing in 30 Years?

The IoE is the most exciting change in our industry since client/server came on the scene in the mid/late 1980s.

Client/server was important because the Internet, Web services, mobile computing are all essentially client/server implementations. Some would argue that Object-Oriented analysis, design and development are significant, but I think these affect only programmers and designers.

Agile is significant – but I think it is an approach, not a technology and anyway, it is transitional. Continuous delivery and DevOps are bringing factory automation processes into software and perhaps are the most appropriate approach for many mobile and IoE implementations. After Waterfall and Agile, Continuous Delivery and DevOps are morphing into 'the third way'.

The IoE, if it develops in the way that some forecasters predict, will affect everyone on the planet. Estimates of the number of connected devices on the IoE range from 50 billion devices to 700 billion devices over the next 5 to 20 years. No one knows of course, but the expectation is that we are on a journey that will increase the scale of the internet by one hundred times. This will take some testing! But how on earth will we approach this challenge?

The Latest Step in the Testing Journey

There have been some quantum leaps in the complexity of our systems and the IoE is the next step on our journey. Let me recount the history of our software testing challenge. I know testing started before we had 'green screens', but it is a convenient starting point.

1. Screen based applications running on dumb terminals were the norm up to the mid-1980s or so (and of course are still present in many companies). Each screen typically had a single entry point

and a single exit point, the content of the screen and data input was limited to text. Screens might be relatively simple – but systems had many screens.

2. With the advent of GUI applications, it became possible to have many windows active at the same time. There were many ways in and ways out. The content of screens could now include graphics, sounds and video. Windows applications had to deal with events – triggered by the user clicking anywhere onscreen or from other windows or applications resident in the GUI environment.
3. At the same time as GUIs arrived, client/server became the architecture of choice for most applications. The scope of an application was no longer limited to workstations but might involve the collaboration of many servers connected with (often flaky) middleware. Performance and reliability become more prominent risks.
4. The emergence of the internet, although hailed as a revolution is really an instance of client/server. What was different though, was the exposure of private networks, functionality and data to the public internet. Security and privacy become much bigger concerns.
5. We are in the thick of the 'Mobile Revolution' right now. The democratisation of software means applications are available anytime and anywhere on varied devices and configurations in numbers far beyond our ability to test comprehensively. The proliferation of apps and devices and our enthusiasm for this expansion knows no bounds.
6. And now, on top of all of the technologies above, we expect that the network of connected 'things' will increase the scale of the internet by perhaps 100 times. These devices range in sophistication from 50 cent components to buildings, ships, cars, airplanes and cities. And that's the point - IoE brings all of these plus 'all of the above' with the full range of sophistication from devices costing a few pennies to the infrastructure of a city that might cost billions.

The IoE brings new levels of complexity and scale. The non-functional risks are reasonably well-known and we know how to address them. *What is new is the need to do functional testing and simulation at scale.*

Some Other Considerations

CMOs May Hold the Budget for IoE

There seems to be a shift in power, at least in the mobile space (which accounts for a significant proportion of the IoE market). It appears that for mobile businesses, the budget for building mobile and IoE systems may be held by marketers. If CMOs hold the purse strings, the justification for IT will be driven by the need to meet both short and long term business goals. Development must align with the need to experiment and learn.

To do this, IT will have to be agile and most likely adopt a continuous or DevOps approach. Delivery will be based on experimentation to drive the rapid evolution of products and services to meet rapidly changing requirements. DevOps, continuous delivery, high levels of automation, simulation, experimentation and test analytics matched with business and production analytics in some blend will be required.

Connecting the Unconnected

The challenge for the network operators around the world and the service providers wishing to gain customers is to connect 'the second half of the world'. In developing economies, it is likely that mobile internet rather than cabled links will be used to connect widely separated communities. Mobile phones took off when the cost of handsets dropped below \$40. Perhaps when smartphones are available at that price, the mobile internet will take off too.

The goal of 'Internet.org by Facebook' [4] is to provide cheaper access to data by (to perhaps 1/100th of the current price) using network extension technologies and unused or white space spectrum [5] and to reduce the volumes of data used through local caching and compression technologies. Connecting the unconnected is happening on a global scale, and a country by country basis and gathering pace. IoE implementations in emerging internet economies may have quite different approaches to local networking than the developed economies.

Recently, the Colombian president announced that every citizen of Colombia now has their own email address. Colombia is an examples of a country where there are 'many connected, many unconnected'. The internet.org service was recently launched in Colombia [6].

The acceptable cost for mobile internet based IoE is very low. For example, the cost of smart meters monitoring energy usage in domestic homes needs to cost no more than a penny per device per month. In the developed economies the costs of mobile internet is still rather too high, but it is coming down.

The constraints of local mobile connectivity in the developing economies will drive the development of cheaper technologies. The developing economies will probably lead the way in implementing the IoE with mobile internet.

What networking technology/protocols will be used to implement IoT local connectivity? One speaker at the MWC suggested that, "No technology dominates yet. We are waiting for technologies to improve in range and performance. The standard technology for connecting the IoT hasn't been invented yet."

We are still at the start of the IoE journey.

Analytics and the People Who Use Them

Most of the data being captured by sensors is time series data, that is, the history of some measurement is captured and stored for a device over a period of time. The device might be fixed or mobile, but ultimately, the data it collects reflects a measurement captured over time. Analytics tools have not been great at integrating and correlating measurements that share different, possibly random capture frequencies. Depending how you cluster and average data, different patterns might emerge.

The data captured ultimately represents some physical attribute or measurement. Capturing data and obtaining statistics is one challenge, but to get value out of the statistics you need also what might be called physics-based modelling in order to get the 'deep insight' and control.

Big Data has been around for a few years now, but the use of analytics or data-science to derive insights from data is still evolving. The discipline of data-science is not yet well-defined and the availability of skilled data-scientists is generally low, but improving. At the same time, however, the data skills of the people who would use the analytics and visualisations are still lacking. Data-driven management as a discipline is still embryonic. One MWC speaker suggested, "You can have the best insight, but if people haven't been trained they'll do what they know, rather than use the data".

The Scope of IoE Testing

The range of concerns that the IoE brings is wider than ever before. Of course, not all IoE systems will be huge, complex and expensive, but most will bring a new technology and risk profile. The approaches we will need to tackle the risks of failure will have to change. Here are my suggested 'dimensions' of the technology and testing problem. This isn't very likely to be the last word.

Scale: The obvious first challenge is scale of course, but that is primarily a logistical problem for implementers. Of course, there will be the scalability challenges at various levels of the architecture and we'll have to do some large-scale load, performance and stress testing.

Hardware-Level functionality: the lowest level devices are sophisticated, but essentially perform simple functions like sensing the value of something or changing the setting, position or speed of a machine. These devices are packaged into objects that will need testing in isolation but most of this will be performed by manufacturers.

Object and Server level functionality: The vast majority of functionality that needs testing will reside on local hubs and aggregators and data-centre-based server infrastructure. Internet-based and native mobile apps will deliver the data, visualisations and control over other aspects of the architecture. Architectures will range from simple web-apps to systems with ten, twenty or more complex sub-systems.

Mobile objects: testing static objects is one thing, but testing objects that move is another. Mobile objects move in and out of the range of networks; they roam across networks. The environmental conditions at different locations vary and may affect the functionality of the object itself. Our sources of data and the data itself will be affected by the location and movement of devices. Mobile devices will drift into and out of our network range, but also drift into and out of other networks, not necessarily friendly ones. Power, interference, network strength, roaming and jamming issues will all have an effect.

Moving networks: Some objects move and carry with them their own local network. A network that moves will encounter other networks that interfere or may introduce a rogue or insecure network into range and pose security problems. Cars, buses, trains, airplanes, ships, shopping trolleys, trash trucks, hot-dog stalls, tractors – almost anything that moves – might carry with them their own networks and bridge to and collaborate with 'friendly' networks as they encounter them. But they must also block foreign and/or unfriendly networks too.

Network security risks at multiple levels: Rogue devices that enter your network coverage area might eavesdrop or inject fake data. Rogue access points might hijack your users' connections and data. Vulnerable points at all levels in your architecture are prone to attack. Networks will need to be hardened and tested.

Device registration, provisioning, failure and security: Devices may be fixed in location or mobile but the initial registration and provisioning (configuration) are likely to be automatic. More complicated scenarios arise where devices move in and out of range of a network or transition between networks. Needless to say, low-power devices fixed in perhaps remote locations are prone to power failures, snow, heat, cold, vandals, animals, thieves and so on. Power-down, power-up and automated authentication, configuration and registration processes will need to be tested.

Collaboration confusion: Mobile, moving devices will collaborate with fixed devices and each other in more and more complex ways and in large numbers. For example, in a so-called Smart City, cars may collaborate as a crowd to decide optimum routes so every car gets to its destination efficiently. But, however these resources are controlled, accidents happen, drivers change their mind, car park spaces will become available and unavailable randomly and so the optimisation algorithm must cope with rapidly changing situations. At the same time, these services must not confuse public services, commercial vehicle drivers, private car drivers and passengers. Managing the expectations of users for all systems that collaborate dynamically will be a particular challenge.

Integration at all levels: Integration of physical devices or software components will exist at every level. Integration of data will encompass the flows of data that is correctly filtered, validated, queued, transmitted and accepted appropriately. Many IoE devices will be sensors chirping a few bytes of data periodically, but many will also be software components, servers, actuators, switches, monitors, trip-ups, heaters, lifts, cars, planes and factory machinery. The consistency, timeliness, reliability and of course safety of control functions will be a major consideration. Industry standards in application domains that are safety-related are likely to have a role. However, for now, much of the legislation and standardisation that will be required does not yet exist.

Big Data – logistics: Needless to say, much of the data collected by devices will end up in a database somewhere. Some data will be transactional, but most of it will be collected by sensors in remote locations or connected to moving objects. A medium sized factory might collect as much as a Terabyte (1,000,000,000,000 bytes) per day. Performance and reliability requirements might mean this data must be duplicated several times over. Wherever it is stored (and for however long) a very substantial data storage service will be part of the system to be tested.

Big-Data – Analysis and visualisation: Analyses of data will not be limited simply to tabulated reports. The disciplines of data science and visualisation are advancing rapidly, but these rely on timely, accurate and consistent data; they rely on data acquisition, filtering, merging, integration and reconciliations of data from many sources, many of which will never be under your control. Often, data will be sparse or collected infrequently or at random times. Data will need to be statistically significant, smoothed, extrapolated and analysed with confidence.

Personal and corporate privacy: The privacy of data – what is captured, what is transmitted, what is shared with trusted or unknown 3rd parties or stolen by crooks and misused is probably peoples' most pressing concern (and this is also a barrier to exploitation of the IoE). The current legal framework (e.g. the Data Protection and privacy laws in the UK) may not be sufficient to protect our personal or corporate privacy. Hackers and crooks are one threat, and central government tracks and listens to them. But they tend to listen to all citizens, not just suspects. Your own government may be seen to be a villain in this unfolding story.

Wearables and Embedded: Right now, there is a heavy focus on wearable devices such as training and activity trackers and heart monitors that connect with apps on mobile phones. Your health data might be integrated with Google Maps to show training routes, for example. Other devices such as smart watches, clothing and virtual reality headsets are available now. But there are increasing numbers of applications where the device is not worn, but are actually embedded in your body.

Healing chips, cyber pills, implanted birth control, smart dust and the 'verified self' (used to ID every human on the planet) are all being field tested. It will be hard not to call human beings 'things' on the internet before too long. Will we need to hire thousands of testers with devices embedded in their body? Surely not. But testing won't be as simple firing off a few messages to servers using the tools we have today. Something much more sophisticated will be required.

Everything connected: The time will come when all of the devices used in a hospital, hotels and factories for example will be tagged or connected. Hospital staff spend a lot of time finding and moving equipment from place to place to hook up to patients and locating equipment faster will save time and could save lives. A less critical application might be in a hotel where every piece of cutlery, cup, plate, dish, glass is tagged and located. How many fewer of these need be bought by a hotel if the location of each is known? The same argument applies to tools, equipment and robots in factories and of course the people who use and maintain them.

The range of issues we need to consider in testing the IoE has increased and the scale of the testing required has increased too.

Part II - Test Strategy for the IoE

The way we have tested in the past, will work for some of the IoE applications of the future. But for many systems, the number of dependent variables that influence their behaviour increases dramatically. The nature of Big Data (think volume, velocity, variety and veracity), merging and integration of multiple data sets means that the range of analyses and possible insights derived are only limited by our imagination.

The change in thinking required to test large IoE implementations is significant. Let's explore some of the gaps in our abilities and how we might approach the challenge.

Modelling, Testing and Test Data

We have some good techniques already to model technical architectures, and tools do exist (used mostly in the systems engineering space) to capture models and to generate code and a covering set of tests automatically. But they tend to be based either on the technical architecture or domain-specific use-cases.

There are very sophisticated real-time modelling and simulation tools used in the automotive and aerospace industries, for example. These model the physical components of cars and planes to enable a large amount of testing to be done without costly full-scale prototypes (which might be tested to destruction). There are proprietary (and even some open source tools) to simulate traffic flows in cities, the movement of people in airports and of course airplanes in the skies.

Sophisticated, expensive simulation tools might be used by pilots, astronauts and Formula 1 drivers. Much better known are the simulators that are used by gamers of course. It is these that might provide the basis of usable tools to simulate large-scale collaboration and scenario testing. The software used to provide a car-chase experience could be used to model traffic flows. Shoot 'em up games could be adapted to simulate the exchange of data or payments rather than bullets, and the social engineering games adapted to simulate the exchange of labour, goods and services. And so on.

But these adaptations are probably not high on many people's agenda. Only large companies like Google, the car and airplane manufacturers, suppliers of power generation and distribution, signaling, traffic control and of those who supply to the military can afford to buy or build such tools. These products are not likely to be available to start-up companies – at least for the next few years.

The best you can do is to run manual simulations supported by tools that can repeatedly generate scenarios to be tested, record the outcomes and replay the simulations for later study perhaps. Needless to say, you need to incorporate test-hooks or other features in your systems and build utilities that will help you to easily inject data, capture and reproduce or replay scenarios.

Cem Kaner has written quite a lot about what he calls 'High Volume Test Automation' [7]. This is a good starting point.

Because the amount of activity and data these systems require and generate, you will be using Big Data test techniques. You'll need to find data that matches your test requirements; you will need to generate, tag, edit and seed data so you can trace its usage; you will need tools to monitor the use of tagged data and the ability to reconcile data from collection, storage, use and disposal. It will be helpful to use the visualisations created in your system for end-users and to enhance them for debugging and diagnostics.

One aspect of test generation worth thinking about is the concept of 'pattern-based test design'. The goal of this technique is to generate tests from known data to simulate particular scenarios that can be characterised as a pattern. Suppose we are testing a self-driving car (Google, are you listening?)

On British roads, where we drive on the left (or should do), cars that turn right and cross the path of oncoming traffic cause a large number of accidents. So an obvious pattern to test would be 'cars turning right that cross oncoming traffic'. If we have a selection of routes with right turns, we could generate a large set of very short journeys that include a right turn. Then we could overlay a second set of journeys that intersect at the junction (to intersect at the same moment) and for each intersecting pair of journeys, run our simulation.

What is the expected result? Well, we would be interested in seeing outcomes where one car or the other is delayed by more than a few seconds perhaps (indecision?) We would look for outcomes where the two

cars appear to occupy the same geolocation at the same time (crash?) and so on. We could run these tests at varying velocities for both the turning and the oncoming cars at many locations across a busy city.

We could, for either car, change the persona of the driver from the automaton to a teenager or retired schoolteacher or change the car itself to a van, lorry or bus and so on. Obviously, we could run simulations with thousands of cars on intersecting journeys simultaneously. Of course, this is a test that we would run eventually – but we would probably not start with that test, as it would be an almost impossible task to debug the failures we would expect to see.

(Given the personas of some drivers I've encountered over the years, I am looking forward to seeing how a Google or other self-driving car copes with Central London traffic or that in Bangalore! Or maybe not.)

At any rate, the goal of pattern-based test design is to feed your automated tools hundreds or thousands of scenarios to simulate. Part of the brief for your developers must be to include features or hooks to facilitate High Volume Automated Testing.

Test Environments; Testing in the Field

Requirements for the scope of test environments for IoE implementations will vary widely. In a simple case, a test lab for a home environment management product could be set up in any office as the scope of the local network is confined to a single household.

In the case of an urban environmental management system that monitors air pollution levels across a city for example, the sensor data capture could be simulated in a lab, but you would expect to have to pilot the service in a real city environment to calibrate the sensors, data aggregation and integration processes and have meaningful data visualisations.

The scale of the field testing of a service that is to be marketed to home owners or the authorities of a large city will vary widely. Field testing to demonstrate a system can accommodate the vagaries of the weather, traffic or the general population to obtain the confidence of stakeholders will become a common (and possibly mandatory) testing stage.

Barcelona has for a long time prided itself in being an innovative city. Recently, the city authorities have established an urban-scale testing facility. The Barcelona Urban Lab [8] has been set up as a test environment for innovative projects that meet the needs (and some other conditions) of the city. Approved projects may use the existing city infrastructure to pilot innovative systems that provide new and useful services to the city. The Urban Lab is specifically positioned as a field test environment to prove the viability of innovative systems.

It is expected that other cities and local authorities around the world will offer similar services and the development of Smart Cities will depend on partnerships between system developers and civic authorities.

Tool Support

There can be no doubt that automated support for testing non-trivial IoE implementations will be essential. A lot of manual and/or ad-hoc testing by developers and testers working closely will be performed, but automated testing will probably dominate. Many of the software components of an IoE infrastructure will provide access to their functionality through APIs or services delivered by web or other emerging messaging protocols so testing with tools should be eased.

Now, these tools may be COTS products or open source but testers may have to create their own test drivers. The technologies used at the local network level (connecting devices/objects to local integrator services) vary and although there are IP- and HTTP-based technologies in use, for which there are

commercial and open source tools, there are quite a few other services that are not well supported by tools yet and custom drivers will be necessary.

The question then arises as to the nature of the tests to be run by these tools. In some cases, the number of tests to be run might be quite small. For example, many mobile apps are quite limited in function and so a combination of manual testing and a number of automated tests through web services or through the GUI of the app itself will suffice.

But there will be circumstances where the number of tests required to demonstrate the functionality and resilience of services will be extremely high. Where there is a high volume of data captured from distributed locations and the patterns of that data are used to make real-time decisions, it would be difficult to test thoroughly without tools and a known set of data (whether real or synthetic) that can be re-used to run series of tests.

It seems inevitable to me that although there will always be a need and opportunity to do manual testing, a much larger proportion of testing will have to be performed by tools than we are currently used to. The tools will need to execute very large numbers of tests. The challenge is not that we need tools to execute tests. The challenge will be, "how do we design the hundreds, thousands or millions of tests that we need to feed the tools?"

So for example, suppose we need to test a smart city system that tracks the movement of vehicles and the usage of car parking in a town. We'll need a way of simulating the legal movements of cars around the town along roads that contain other cars that are following their own journey. The cars must be advised of their nearest car space and the test system must direct cars to use the spaces that they have been advised of. We could speculate on some heuristics that guide our test system to place cars where their owners want them. But.... life happens and our simulation might not reflect the vagaries of the weather, pedestrians, drivers and the chaotic nature of traffic in cities.

Not every system will be as complex as this. But the devices now being field tested in homes, hospitals and public places all have their nuances, complications and will experience unexpected events. Even a 'simple' healthcare application that warns a patient of forthcoming doctor's appointments, schedules prescribed drugs and monitors your symptoms or vital signs could trigger hundreds or thousands of scenarios.

Increasingly, small simple systems will interact with other systems and become more complex entities that need testing. It's only going one way.

Test Analytics, Visualisation and Decision-Making

I have written about Test Analytics before [9].

Increasingly, practitioners are looking for automated ways to do things. Behaviour-Driven Development connects requirements, test code and code itself by generating and executing feature-level tests. Automated build and test processes create event and test outcome data that potentially can be used to trace progress, coverage and trends in your test results.

Think of your DevOps processes as 'things' that support the evolution, maintenance and smooth running of your production systems. These processes (or at least the data they produce) are part of the system deliverable itself. These automated processes exercise your systems and monitor their vital signs just like sensors. In your production systems, the analytics provide valuable usage and market information. This same instrumentation can inform your development processes in a similar way and statistics from your DevOps processes can be designed to tell you how your development process is working.

If marketers are in the driving seat, and experimentation in production is an explicit goal of a system, then developers will quickly become expert in instrumenting components to capture the required analytics.

Of course analytics capture data to track user behaviour, but it will be easy to extend that to capture other data of interest to testers and developers.

If instrumentation can be enhanced to capture data that self-checks or reconciles selected outcomes, then you could envisage having a sensor-network on your production code. The analytics so derived can be set up as 'trip-wires' that signal problems in production. Of course, this same code can operate in your DevOps environment and give testers an indication of the health of your system in the test lab.

If your production systems are platforms for experimentation, then the code that serves marketers can also serve developers and testers. Think of your analytics code as your sensor network. DevOps processes are things, too.

Performance Testing and Test Data

Network protocols that are IP based, for example XMPP [10] and MQTT [11] might be used as a local aggregators or as central hubs working at high volumes. There are few dedicated tools to support the simulation of loads but custom drivers/utilities are quite straightforward to create. Also, cloud-based services are becoming available allowing developers to outsource these aspects of their architectures. We've built an MQTT broker and MongoDB service ourselves [12].

Other Low-Power and Lossy Network (LLN) protocols such as ZigBee [13], 6LowPAN [14], DASH7/RFID [15], Bluetooth [16] and NFC [17] only work at the local level. There are few tools that support load testing of these protocols, but if you do need to simulate hundreds or thousands of remove sensors and objects, then crafting an effective driver to plug into existing performance toolkits should not be so hard. After all, the drivers might only need to reproduce the regular chirps of sensors.

What makes life harder is when the data that is captured by sensors must be coherent. For example, if you need to chirp location data for a car, a random set of location co-ordinates will simply not do. Cars do not normally zip around randomly (and at very high speed). So your driver may have to use a traffic route using the Google Maps API and the periodic chirps of data calculated to simulate a real car travelling at a reasonable, defined speed along the route.

If you need to track the movement of devices carried by people in a shopping mall, then perhaps you will have to capture the GPS data using a tool. I'm currently experimenting with a tool called GPS Logger for Android [18] to do just this. To create a variety of unique paths, you could edit together segments of intersecting paths, for example. Of course, if your app collects data such as this, you could use an early version to collect some test data for you. Once in production, you will have a plentiful supply of data to use in testing (when appropriately anonymized).

New Model Testing

Why do we need a 'New Model for Testing'?

The current perspectives, styles or schools of testing will not accommodate emerging approaches to software development such as continuous delivery and, the new technologies such as Big Data, the Internet of Things and pervasive computing. These approaches require new test strategies, approaches and thinking. Our existing models of testing (staged, scripted, exploratory, agile, interventionist) are mostly implementations of testing in specific contexts.

Our existing models of testing are not fit for purpose – they are inconsistent, controversial, partial, and often proprietary and stuck in the past. They are not going to support us in the rapidly emerging technologies and approaches of the IoE.

I have proposed an underlying model of testing that is context-neutral and I have tried to shed some light on what this might be by postulating the Test Axioms, for example [19, 20]. The Axioms are an attempt

to identify a set of rules or principles that govern all testing. Some testers who have used them think they work well. They don't change the world, they just represent a set of things to think about – that's all. But, if you choose them to be true, then you can avoid the quagmire of debates about scripted versus unscripted testing, the merits and demerits of (current) certifications or the value of testing and so on.

The New Model for Testing [21] is an extension to this thinking. The model represents the thought-processes that I believe are going on in my own head when I explore and test. You might recognise them and by doing so, gain a better insight into how you test too. I hope so. As George Box said, 'essentially, all models are wrong, but some are useful'. This model might be wrong, but you might find it useful.

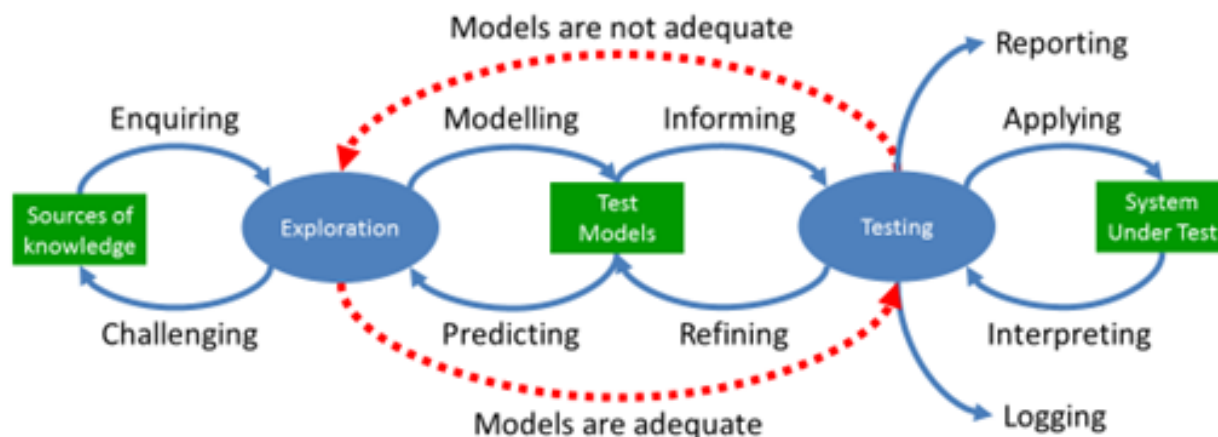
The New Model of Testing attempts to model how testers think and you can see a full description of the model, the thinking behind it and some consequences here: <http://dev.sp.qa/download/newModel>

Ignore Test Logistics

When tests are performed on-the-fly, based on mental models, your thought processes are not visible to others; the thinking might take seconds or minutes. At the other extreme, complex systems might have thousands of things to test in precise sequence, in complicated, expensive, distributed technical environments with the collaboration of many testers, technicians and tool-support, taking weeks or months to plan and apply.

Depending on the approach used, very little might be written down or large volumes of documentation might be created. I call the environmental challenges and documentary aspect 'test logistics'. The environmental situation and documentation approach is a logistical, not a testing challenge. The scale and complexity of test logistics can vary dramatically. But the essential thought processes of testing are the same in all environments.

So, for the purpose of the model, I ignore test logistics. Imagine, that the tester has a perfect memory and can perform all of the design and preparation in their head. Assume that all of the necessary environmental and data preparations for testing have been done, magically. Now, we can focus on the core thought processes and activities of testing. Here is the model:



The model assumes an idealised situation (like all models do), but it enables us to understand more clearly about what testers need to think about.

At the most fundamental level, all testing can be described in this way:

1. We identify and explore sources of knowledge to build test models
2. We use these models to challenge and validate the sources of knowledge
3. We use these models to inform (development and) testing.

I make a distinction between exploration and testing. The main difference from the common view is that I will use the term Exploration to mean the elicitation of knowledge about the system to be tested from sources of knowledge.

By excluding the logistical activities from the New Model, then the processes can be both simplified and possibly regarded as universal. By this means, perhaps the core testing skills of developers and testers might coalesce. Testing logistics skills would naturally vary across organisations, but the core testing skills should be the same.

From the descriptions of the activities in the exploration and testing processes, it is clear that the skills required to perform them are somewhat different from the traditional view of testing as a staged activity performed exclusively by independent test teams. Perhaps the New Model suggests a different skills framework. I have put together a highly speculative list of skills that might be required.

I hope the model stimulates new thinking and discussion in this field.

The Future for Testing and Testers

My suggestions below are partly informed by what I have seen in the technology and the testing markets and friends and colleagues who have shared their experiences with me.

Manual v Automated Testing?

It seems to me that high levels of test automation are bound to be required to make the IoE a reality. Automation will not make testing easy; it will make testing possible.

Web services, as an example, might be simple to test, but can also be devilishly complex but we have to use some form of driver to test them. The distinction of 'manual' and 'automated' testing in terms of ease, effectiveness and value is fatuous – we have no choice in the matter. What I have called the 'application of tests' in the New Model may be performed by tools or people, but ONLY people can interpret the outcomes of a test. That is all there is to it.

But I should add that more and more, test automation in a DevOps environment is seen as a source of data for analysis just like production systems so analysis techniques and tools are another growing area. I wrote a paper that introduces 'Test analytics' here [9]. This is a field that testers should pay attention to, I think. Given the shortage of data scientists out there, it is a field that might pay well.

Should testers learn how to write code?

I have a simple answer – yes.

Now it is possible that your job does not require it. But the trend in the US and Europe is for job ads to specify coding skills and other technical capabilities. More and more, you will be required to write your own utilities, download, configure and adapt open source tools or create automated tests or have more

informed conversations with technical people – developers. New technical skills do not subtract from your knowledge, they only add to it. Adding technical skills to your repertoire is always a positive thing to do.

If you are asked to take a programming or data analytics course, take that opportunity. If no one is asking you to acquire technical skills, then suggest it to your boss and volunteer.

Shift-Left

It seems like every company is pursuing what is commonly called a 'shift-Left' approach. It could be that test teams and testers are removed from projects and developers pick up the testing role. Perhaps testers (at least the 'good' ones) are being embedded in the development teams. Perhaps Testers are morphing into business or systems analysts. At any rate, what is happening is that the activities, or rather, the thinking activities of testing are being moved earlier in the development process.

This is entirely in line with what testing leaders have advocated for more than thirty years. Testers need to learn how to 'let go' of testing. Testing is not a role or stage in projects; it is an activity. Shift-left is a shift in thinking, not people. I suggest that testers view this as an opportunity, rather than a threat.

'Test early, test often' used to be a mantra that no one followed. It now seems to be flavour of the month. My advice is don't resist it. Embrace it. Look for opportunities to contribute to your teams' productivity by doing your testing thinking earlier. The left hand side of the New Model identifies these activities for you (enquiring, modelling, challenging and so on). If your test team is being disbanded, look at it as an opportunity to move your skills to the left.

There is something of a bandwagon for technical people advocating continuous delivery, DevOps and testing/experimenting in production. It seems hard for testers to fit into this new way of working, but again, look for the opportunity to shift left and contribute earlier. Although this appears to be a technical initiative, I think it is driven more by our businesses. I learned recently that marketing budgets are often bigger than company IT budgets nowadays. Have a think about that.

Marketers may be difficult stakeholders to deal with sometimes, but their power is increasing, they want everything and they want it now. The only way to feed their need for new functionality is to deliver in continuous, small, frequent increments. If you can figure out how you can add value to these new ways, speak up and volunteer. Of course large, staged projects will continue to exist, but the pressure to go 'continuous' is increasing and opportunities in the job market require continuous, DevOps and analytics skills more and more. Embrace the change, don't resist it.

I wish you the best of luck in your leftwards journey.

Summary

When we moved from testing dumb-terminals to GUIs, the number of possibilities exploded and it took time to get the hang of it. It was like learning to test in three dimensions rather than two. Testing the Internet of Everything requires a similar increase in scale, diversity and complexity. Testing in the fourth dimension, perhaps?

Although at the scale of a single component or sub-system testing is pretty much the same as before, for system testing we need to adopt simulation methods using high volume test automation. The tools we will need to do this probably don't yet exist so the pioneers in this space will have to customise performance and functional test tools and write their own drivers for a while.

High volume test automation requires test models, test data generators and automatic oracles. Modelling, simulation, analytics, visualisation and tool-supported decision-making will become important capabilities of test architects and testing teams. Testers used to (so-called) manual testing will have to learn how to create better test models and how to use them with more technical modelling and simulation tools.

Obtaining trustworthy, accurate test environments and meaningful test data will, as always, cause big headaches for testers.

Large-scale test environments in the lab and in the field will be required and the boundaries between experimentation in production and testing in the lab will become blurred. Test Analytics derived from DevOps processes will become a critical discipline in testing the Internet of Everything.

Some new test approaches will be required and the New Model is an attempt to trigger new thinking in this area.

[Back To Index](#)

References

1. The Internet of Everything – What is it and how will it affect you?, Paul Gerrard, <http://gerrardconsulting.com/sites/default/files/IOEWhatIsIt2.pdf>
2. Internet of Everything – Architecture and Risks, Paul Gerrard, <http://gerrardconsulting.com/sites/default/files/IOEArchitectureRisks.pdf>
3. Mobile World Congress, Barcelona, 2-5 March 2015, <http://www.mobileworldcongress.com/> .
4. Internet.org by Facebook, <http://internet.org/>
5. White Space Spectrum, [http://en.wikipedia.org/wiki/White_spaces_\(radio\)](http://en.wikipedia.org/wiki/White_spaces_(radio))
6. Internet.org App Launches in Colombia, <http://newsroom.fb.com/news/2015/01/internet-org-app-launches-in-colombia/>
7. An Overview of High Volume Test Automation, Cem Kaner, <http://kaner.com/?p=278>
8. Barcelona Urban Lab, <http://www.22barcelona.com/content/view/698/897/lang,en/>
9. Thinking Big: Introducing Test Analytics, Paul Gerrard, <http://www.gerrardconsulting.com/sites/default/files/TestAnalytics2.pdf>
10. XMPP, The Extensible Messaging and Presence Protocol , <http://xmpp.org/about-xmpp/>
11. MQTT, " a machine-to-machine (M2M)/"Internet of Things" connectivity protocol, <http://mqtt.org/>
12. Gerdle, Gerrard Consulting, <http://gerdle.com>
13. ZigBee, <http://en.wikipedia.org/wiki/ZigBee>
14. 6LowPAN, IPv6 over Low Power Networks, <http://datatracker.ietf.org/wg/6lowpan/documents/>
15. DASH7, RFID standard, <http://en.wikipedia.org/wiki/DASH7>
16. Bluetooth, , <http://en.wikipedia.org/wiki/Bluetooth>
17. NFC, Near Field Communication, http://en.wikipedia.org/wiki/Near_field_communication
18. GPS Logger for Android, <http://code.mendhak.com/gpslogger/>
19. Test Axioms, Paul Gerrard, <http://testaxioms.com>
20. The Tester's Pocketbook, Paul Gerrard, <http://testers-pocketbook.com/>
21. A New Model for Testing, Paul Gerrard, <http://dev.sp.qa/download/newModel>



Paul Gerrard is a consultant, teacher, author, webmaster, developer, tester, conference speaker, rowing coach and a publisher. He has conducted consulting assignments in all aspects of software testing and quality assurance, specialising in test assurance. He has presented keynote talks and tutorials at testing conferences across Europe, the USA, Australia, South Africa and occasionally won awards for them.

Educated at the universities of Oxford and Imperial College London, in 2010, Paul won the Eurostar European Testing excellence Award and in 2013, won The European Software Testing Awards (TESTA) Lifetime Achievement Award.

In 2002, Paul wrote, with Neil Thompson, "Risk-Based E-Business Testing". In 2009, Paul wrote "The Tester's Pocketbook" and in 2012, with Susan Windsor, Paul co-authored "The Business Story Pocketbook". He is Principal of Gerrard Consulting Limited and is the host of the UK Test Management Forum and the UK Business Analysis Forum.

Mail: paul@gerrardconsulting.com | Twitter: @paul_gerrard | Web: gerrardconsulting.com

The Tester's Kaleidoscope to Internet of Things

- by **Diwakar Menon**

Our tomorrow is on us today!

We are feeling the initial effects of that inter-connected world, where our watch nags us into doing things – be it waking up or exercising or going off for meetings. I used to be called a beep driven life form at one time, since my alarm on my old Casio organizer would go off, and I would get up, pick up the notes, and head for a meeting. Since then, things have seemingly got worse!

So in a lighter vein, soon, my microwave, will decide what I eat, and it has an able ally, in my refrigerator!

My wardrobe will tell me what to wear, depending on my schedule, the weather, etc.

My home may decide to lock me in, if I am not out in time!

My car will decide where I need to go, depending on my appointments

My mobile will decide to book a dinner for two with my wife, since it is quite likely I have forgotten it is our anniversary

My table will decide what plates I should eat in depending on what the microwave and refrigerator have decided.

My only hope is I hope I can decide what I do in bed!

So welcome to this ubiquitous connected world, where the centre of attention is ME! However, to deliver that experience, and to test whether that experience is delivered, we will need to think about the Internet of Things from a variety of perspectives.

The Human Experience is what will really matter.

It's all going to be about what I experience, based on what I share (or what it gleans from what I have shared). I will be at the centre of this inter-connected world, as a user, where sensors, gadgets, gateways, clouds, applications, profilers, and big data crunchers collide.

The Human Experience, when you look at it from the Internet of Things (or Internet of Everything!) perspective, is being delivered as a sum of multiple layers interacting with each other:

- From the sensors
- To the gateways
- To the storage
- To the applications

The experience will need to be tested as a sum of all of the moving parts.

Let's then look at the ways by which Human Experience can be possibly defined?

- It should simplify life and remove the tedium from the tasks I do
- It should enable me as a user by allowing me control
- It must be aware of me as a user
- It must be aware of the context in which I will be using it

All of these need to work along the different layers we stated. So as a tester, I need to be aware, not just of the experience itself, but how the underlying parts move in tandem to deliver the experience.

We will need to worry about it from a technical perspective.

At another dimension, we will need to focus on different aspects, each of which opens up a lot more possibilities

- The Structure – which represents my integrated environment and all the moving parts
- The Function – which represents what the devices, the cloud, the applications do. Examples are, the start and stop, the basic functions, the algorithms that drive the experience
- The Data – which represents the aggregation of data that the different layers will collate, its correctness, its security, etc.
- The Timing – how these different functions need to perform and any characteristics- specific to the performance

- The Platform - the complex world of Operating systems, tools, devices and its associated interconnectedness
- The Usage - the patterns of usage, Using analytics to drive usage, the environment of its use, etc.)

We will need to worry about the Quality Criteria

Another dimension we would need to keep in mind, is the set of Quality Criteria that we will need to validate the entire experience against. Viz;

- Interoperability
- Connectivity
- Functionality
- Recoverability
- Reliability
- Compliance
- Security
- Installability
- And of course the User Experience in putting this all together!

Some other considerations

If, and only if, the world provided us time to do these things! Time apart, it will throw other challenges at us

- Fragmentation

Be it the IoT platforms, the hardware involved or the embedded software, there is simply too much fragmentation. Multiple devices, multiple protocols, multiple applications, multiple ways by which data is sliced, diced and depicted

- Diversity

With diverse platforms, gateways, plethora of devices being built everyday keeping pace with diversity is a big challenge

- Complexity

Obviously, knowing what my refrigerator has, what my microwave wants to do, and the day of the week when I prefer my eggs poached or fried, there is an element of complexity in being context aware and context sensitive

- Rapid and shorter testing cycles

Our customers are not going to wait long to figure out when they can launch the new hot product that integrates security devices to guest lists, or finds out that the bulk of my friends prefer the traditional south Indian cuisine to the continental. They will want to get to market fast, take advantage of the first mover advantage. This, in turn means an increased pressure on us to deliver!

- Lack of standards / protocols

Google Thread, AllJoyn / AllSeen Alliance, Intel, Qualcomm etc.

- Knowledge of tools and complex IoT ecosystem

Oh! And did I forget to mention that we need to be aware of all of these various things at the drop of the hat (before my broom discovers that it is there and sweeps it away)

So, in summary, test & development will most often happen in quick cycles, with Continuous Integration/Continuous Development and other concepts coming into play, coupled with tools that support methodologies like behaviour driven development.

Our approach to testing will take on different ways of looking at testing, from risk driven; context driven; heuristic driven; pattern driven to change driven. **And at the epicentre will be the human tester, who has to provide a wide ranging opinion on the goodness of the release to his stakeholders!** Tester skills will be challenged with their abilities to play the game at multiple levels.



Diwakar is the founding director of Last Mile Consultants Technology Solutions Pvt. Ltd. Prior to founding Last Mile, Diwakar has worked in senior management roles in global organisations like Tech Mahindra, Perot Systems (now Dell), Deutsche Software. He has also been a key part of strategic engagements in Tata Unisys (now TCS) and CMC Ltd.

Diwakar has over 25 years of delivery and consulting experience ranging from test consulting to pre-sales to test delivery. He also has exposure to a wide variety of domains, from Healthcare, Telecom, Capital Markets and Travel & Transportation. He has pioneered and set up end-to-end test services across multiple lines of business, consolidating the services across the organisation. Diwakar has also led consulting engagements directly with end customers, to enable them transform their test organisations driving process improvements, cost transformations and enabling shared services.

Love to Write?



Write For Us

Your ideas, your voice. Now it's your chance to be heard!

Send your articles to editor@teatimewithtesters.com

In the school of Testing

for your better learning & sharing experience

Open Source Tools for Networking Devices Validation



- by **Sujata Verma**

A product can be anything; it can be hardware or a piece of software. In order to test a product we should know or at-least try to find answers to some of the general questions like what is the purpose, the end user, the requirements, the features and existing issues. Networking devices are products which connect the digital world and if not properly tested, may result in loss of data and connectivity. The networking devices include routers, bridges, switches, hubs, backhaul devices including the hybrid devices like proxy server, modems, wireless access points etc.

Types of Traffic

To test networking devices we need to understand the traffic which can be sent through the devices. Traffic can be divided broadly into three types: unicast, multicast and broadcast. Unicast traffic refers to the traffic which is sent directly to the intended device. Multicast refers to the type of traffic which is intended or can be sent to group of devices. Broadcast traffic is the traffic which is sent to all the devices in the network.

There are many different types of free open source tools available, which can be used to simulate or generate real time network traffic. Networking devices are primarily involved in the routing, switching or filtering the packet between the hosts. Tools simulate the packets and the network path to be tested can be between end-to-end hosts or the test target can be the stack of the device. Each tool has to be selected based on functionality to be tested and should be used properly.

This article explains basic open source tools and commands which can be used to test the behaviour of networking devices.

Measuring Reach-ability

Ping is the tool which is commonly used to test the communication between the two devices. The "ping" utility offers many options and these options differ with operating system.

Any Linux based PC (I have used Fedora core 14 to verify the commands) can be used to try these commands.

To generate unicast/multicast/broadcast packets:

```
ping "host ipaddress"  
ping -f -L "multicast-address"  
ping -f -b "network-address"
```

Eg.

```
ping 192.175.1.1  
ping -f -L 224.10.10.10  
ping -f -b 192.175.1.255
```

Another useful option is "-p" which can be used with "ping" command, it specifies the pattern i.e. the specific data pattern to be sent in the packet:

```
ping -p "pattern" "host-ipaddress"  
ping -p ff 192.175.1.1
```

Throughput Validation

Throughputs of networking device can be measured with another open source tool known as "Iperf". It is used to determine the maximum achievable bandwidth on IP networks. Iperf has two versions: iperf2 and the latest version known as iperf3 which is a redesign of the original version and is not backward compatible.

To install iperf3 download it from

<http://downloads.es.net/pub/iperf/iperf-3.0.6.tar.gz>

Use the following commands to install it in Linux server and clients:

```
tar -xvzf iperf-3.0.6.tar.gz  
cd iperf-3.0.6  
./configure  
make  
make install
```

Now iperf3 can be used on command line.

a) To know udp/tcp throughput use following commands:

Server side:

```
iperf3 -s
```

Client side default iperf3 will run in TCP mode

```
iperf3 -c <serveripaddress> -t 60 -i 10 -P 4
```

For udp "option - u" is to be used but pair has to be increased as per the throughput of the link. We can use another option "-b" to specify the target bandwidth, for TCP the bandwidth used is unlimited but for UDP default 1 Mbits/sec is used. To increase this limit use option as given below:

```
iperf3 -c <serveripaddress> -u -b 100M -t 60 -i 10
```

Another option which affects throughput and jitter value is read/write buffer size. It can be defined using "-l" option and defaults to 8KB for udp. If the read/write buffer size is smaller the jitter value will be less but as you increase this value, the jitter will get increased even though the throughput change will be minimal.

```
iperf3 -c <serveripaddress> -u -b 100M -t 60 -i 10 -l 1KB
```

```
iperf3 -c <serveripaddress> -u -b 100M -t 60 -i 10 -l 8KB
```

Ethernet Traffic Generator

PackEth is a tool which can be used for creating and sending any type of data packet. It has a builder interface to create any Ethernet based OSI layer 2, layer 3 and layer 4 packets like UDP, TCP and IGMP. It can also be used to test jumbo packets. It has both windows and Linux versions and can be downloaded from <http://packeth.sourceforge.net/packeth/Home.html>.

File Help

Builder Gen-b Gen-s Pcap Load Save Default Default Interface Send Stop

Link layer

☐ ver II

☒ 802.3

☐ 802.1q

MAC Header

Destination Select

Source Select

Length 0x ☒ auto

802.1q VLAN fields

☐ QinQ 0x8100 0x0000

Tag ID 0x8100

Priority 0 (Best effort)

☐ Cfi VLAN ID 0x001

802.3 LLC field values

Type ☒ LLC ☐ LLC-SNAP

DSAP 0xAA SSAP 0xAA

Ctrl 0x03 OUI 0x

PID 0x0800 IPv4

Next layer ----> ☒ IPv4 ☐ Arp packet ☐ User defined payload

IPv4 data

Version 0x4 Header length 0x5 TOS 0x00 Select Total length ☒ Auto Identification 0x1234

Flags 2 Select Fragment offset 0 TTL 255 Protocol 17 Reserved Header cks 0x ☒ Auto

Source IP Select Destination IP Select Options 0x

Next layer ----> ☒ UDP ☐ TCP ☐ ICMP ☐ IGMP ☐ User defined payload

Source IP Select Destination IP Select Options 0x

Next layer ----> ☒ UDP ☐ TCP ☐ ICMP ☐ IGMP ☐ User defined payload

UDP data

Source port Destination port Length ☒ Auto Checksum 0x ☒ Auto

☐ Udp payload 0x

Pattern:

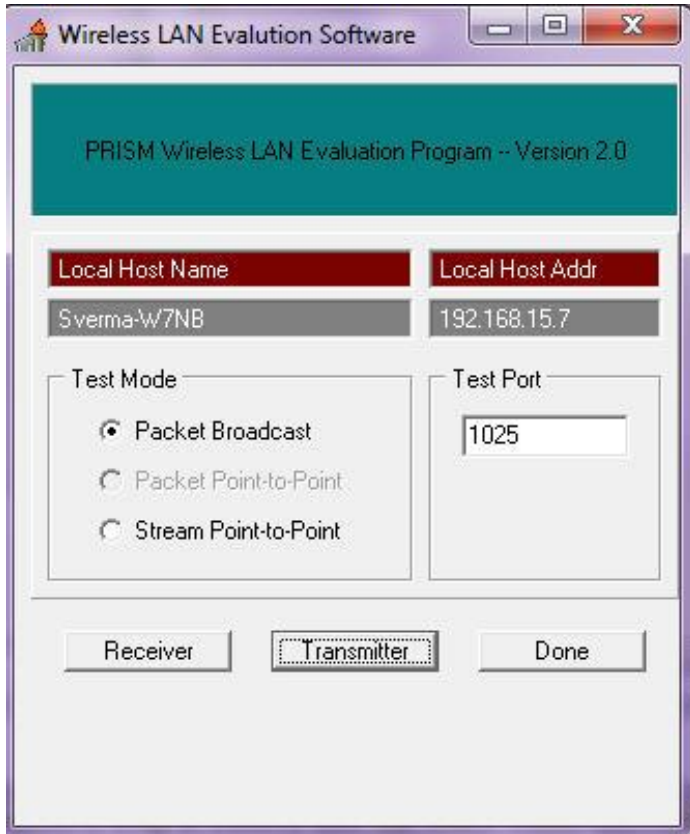
Length:

Apply pattern

Select payload

Broadcast traffic Generator-LANEVAL:

Broadcast traffic can create havoc in the network. All networking devices should be capable of handling broadcast traffic. One simple ARP request packet can create thousand on instances if loop is formed and can make whole network down within few minutes if not handled properly. Harris Semiconductor developed LANEVAL program to test Harris's 11-mbit/s Prism radio chipset, but it can be used to test bandwidth of any network device as it generates thousands of broadcast packets per second. It can be downloaded from <http://www.ke5fx.com/uwave.html> by searching the keyword "LANEVAL".



Conclusion:

Testing a networking device can be tricky and should be tested carefully before deploying them to field. The above tools can be considered as basic steps towards meeting functional expectation and performance as per the requirement.



Sujata K Verma is in System QA and embedded testing domain for close to 12 years and working as QA Manager in Proxim wireless, Hyderabad, India

She can be contacted over [LinkedIn](#) or via email on sujatav@gmail.com



Road to Test Automation

- Part 5

- by Georgios Kogketsof

Sizzle in Webdriver

With my test team when we started developing the automation test scripts the choice we made for tooling was Selenium-RC along with Xpath for writing our locators. This set of tooling served us well for number of years and when the time was right we took the decision to switch to Webdriver and CSS locators.

Because we did not wanted to make both changes in one refactoring we start converting the Xpath to CSS locators and it was a breeze.

What made the transition to CSS so easy was that in Selenium-RC Sizzle was extensively used to "extend" the normal CSS selector lingo with more advanced spices. When we switched to Webdriver the first thing we discovered was that Sizzle was only injected if the browser did not support native CSS selectors unfortunately we were mainly working with Firefox for which the Webdriver uses native CSS and all of our extended CSS locators failed.

To solve the above problem we went back to Webdriver fundamentals and try to alter its API; Webdriver uses the "By" class to encapsulate all possible ways of locating an element. This shows nice in your code, because you can have nice little snippets that make sense as shown in Figure 1:

```
ExpectedConditions
    .visibilityOfElementLocated(
        By.cssSelector(
            "css=a.confirmation_link:not(.hidden)")) ;
```

FIGURE 1 FIND BY SNIPPET

Of course, the above is a Sizzle CSS locator so it would not work in native CSS. Studying the Webdriver API for "By", we see the class itself is a holder for implementing classes of itself - a nice escape from the typical programming practices. We decided that this class should support Sizzle, and to do so, we extend it, initially to add our hook in the cssSelector method as shown Figure 2:

```
public abstract class ByExtended extends By {
    public static By cssSelector(final String selector) {
        if (selector == null)
            throw new IllegalArgumentException(
                "Cannot find elements when the selector is null");
        return new ByCssSelectorExtended(selector);
    }
}
```

FIGURE 2 BY CLASS EXTENSION

We've basically extended the inner class inside By, named ByCssSelector. Basically we used the existing code, only to check if it actually matches something. The code has a nice "habit" of throwing an exception if it does not, so we cleverly "wrapped" it to do our biddings (for brevity, I only give you the code for the 1st method ;) as shown in Figure 3:

Key here is our use of InvalidElementStateException - it is the exception thrown by Webdriver if the CSS cannot be located. We're not dealing with the other possibilities thus trying to make this extension function as the original in all other cases.

```

public static class ByCssSelectorExtended extends ByCssSelector {
    private String ownSelector;
    public ByCssSelectorExtended(String selector) {
        super(selector);
        ownSelector = selector;
    }

    @Override
    public WebElement findElement(SearchContext context) {
        try {
            if (context instanceof FindsByCssSelector) {
                return ((FindsByCssSelector) context)
                    .findElementByCssSelector(ownSelector);
            }
        } catch (InvalidElementStateException e) {
            return findElementBySizzleCss(ownSelector);
        }
        throw new WebDriverException(
            "Driver does not support finding an element by selector: "
            + ownSelector);
    }
    // ... ommitted code ...
}

```

FIGURE 3 EXCEPTION THROWING

The provided solution has three major steps

- Try the *failed CSS locator* with Sizzle (you may want the opposite, *Sizzle first then native CSS*)
- Check if Sizzle exists in the page, if not inject Sizzle so this may work
- Try via Sizzle the failed css locator.
-

To do so, we want two helpers: `isSizzleLoaded()` and `injectSizzleIfNeeded()`, both using the Webdriver API to execute stuff via the `JavascriptExecutor`. Having those, the method `findElementBySizzleCss` is as simple as show in **Error! Reference source not found.** below:

```

public WebElement findElementBySizzleCss(String cssLocator) {
    injectSizzleIfNeeded();
    String javascriptExpression = "return Sizzle(\"" + cssLocator + "\")";
    List elements = (List)
        ((JavascriptExecutor) getDriver())
            .executeScript(javascriptExpression);
    if (elements.size() > 0)
        return (WebElement) elements.get(0);
    return null;
}

```


In conclusion, going back to our original example, here it is using our "Extended" version of By as shown in Figure 4:

```
ExpectedConditions
    .visibilityOfElementLocated(
        By.cssSelector(
            "css=a.confirmation_link:not(.hidden)") ) ;
```

FIGURE 4 BY EXTENDED VERSION

The last major issue to overcome was that when a wrong Sizzle selector is provided (one that is syntactically wrong) the Sizzle engine returns *null*. However, when trying to log this via an exception logger, Webdriver receives an NPE trying to output which selector and type was used; we never did set this up! The exception logged is the NPE and not the actual exception about the missing/deformed selector, causing confusion. To fix this, since the API was protected we did the following as shown in Figure 5:

```
private void fixLocator(SearchContext context, String cssLocator, WebElement
element) {
    if (element instanceof RemoteWebElement)
        try {
            Class[] parameterTypes = new Class[] { SearchContext.class,
                String.class, String.class };
            Method m = element.getClass().getDeclaredMethod(
                "setFoundBy", parameterTypes);
            m.setAccessible(true);
            Object[] parameters = new Object[] { context,
                "css selector", cssLocator };
            m.invoke(element, parameters);
```

FIGURE 5 FIXLOCATOR SNIPPET

As you can see from the extract of code above, we have to modify the accessibility of "setFoundBy" so we can invoke it. After we do that, the call (using the Reflection API) will succeed and hence, no NPE anymore.

So that was all I had in my list to share with you all. Let's keep in touch if you would like to discuss more on this subject.

Note: This article originally appeared in <http://seleniumtestingworld.blogspot.gr/2013/01/adding-sizzle-css-selector-library-and.html>



Georgios Kogketsof is a full time test engineer. In his 15 years of testing experience has worked in multinational, multicultural testing projects for major corporations in the defense industry and in digital marketing. George's expertise as a developer and as a tester fused in creating a hybrid model of a test engineer proved to be extremely effective in automation testing. Over the years George and his testing team have developed a methodology for creating fast, maintainable scripts for automation testing web apps.

George is proud in his involvement in the creation of the open source testing framework Stevia.

Visit his blog at: <http://seleniumtestingworld.blogspot.gr/>

Get Stevia at: <https://github.com/persado/stevia>



ENJOY OUR STUFF?
LIKE US ON FACEBOOK!

facebook.com/TtimewidTesters

Bringing Manual and Automated Testing Together



- by Cullyn Thomson

Looking back on the [history of software testing](#), automated testing isn't actually brand new.

In fact, [James Bach](#) has written that the practice of test automation predates even the concept of "dedicated software testers":

*Test automation is not at all new. What's comparatively new is the idea of a **tester**. Long ago, in the late 40's, dedicated testers were almost unknown. Programmers tested software. Throughout the sixties, papers about testing, such as the proceedings of the IFIPS conferences, almost always assumed that programmers tested the software they built. Testing was often not distinguished from debugging. As larger and more complex systems arose, the idea of dedicated software testers came into vogue.*

Thinking in extremes

And yet even though test automation itself isn't new, it has become quite a hot topic very recently (possibly bolstered by the continuing spread of development methodologies and strategies like [Agile](#) and ATDD/TDD).

The conversation is dramatically polarized. Those at one extreme vehemently oppose automation, seeing it as a death sentence to good ol' fashioned manual testing, human testers, and even software testing in general. Those at the other extreme blindly champion automated testing, dismissing exploratory testing as "playing around."

This “Us vs. Them” view is producing a mentality that says automated and manual testing are at irreconcilable odds with one another, that they can’t get along, that you can only use one **or** the other. And that’s a shame, because “or” sure is limiting.

“And” instead of “or”

So instead of “or,” if you really care about your application’s quality and integrity, I urge you to consider “and.”

Don’t settle for using only one method of testing. Using both automated and manual testing allows you to reap the benefits of both techniques while mitigating the shortcomings of using only one or the other.

Here are two examples:

- Automated testing can help you speed up testing cycles, test constantly, and test more consistently, but at the end of the day, an automated script can only check what you tell it to check. Testing faster is good, but testing less thoroughly is bad.
- Manual testing – particularly exploratory testing – can unveil lots of problems, from user experience issues to serious bugs that no one even considered, but doing it well takes a long time and all of the tester’s attention. When you have to do everything by hand, you simply can’t do as much.

Manual testing and automated testing don’t diminish one another; they **enhance** one another.

Making it work for you

Whether your company currently is purely an automation shop or relies solely on manual testing, you can make this magical testing combination work for you.

Introducing manual testing

If your team has been neglecting manual testing, believing that your automated tests provide all the coverage you need, think again.

Re-evaluate your automated suite to make sure that your automated tests are as robust as possible and that they’re truly exercising your application. Likewise, review the results of automated test runs with a critical eye. Remember that [a 100% pass rate doesn’t necessarily mean your site is bug-free](#); it could mean that your automated tests aren’t well-written or aren’t doing what you think they’re doing. Then update your automated scripts as needed to get them in the best condition possible.

Once you’re confident that your suite is up to date, it’s time to turn to manual testing. Assign test cases that aren’t suited to automated testing to your test team for manual scripted testing, and be sure to give your testers time for exploratory testing. Doing so may involve a shift in priorities and testing schedules, and that’s OK.

The important thing is that by bringing manual testing into the mix, you’ll be able to cover your application’s functionality even more thoroughly.

Introducing automated testing

On the other hand, if your team has been using only manual testing, add automated testing into the mix.

To start, you'll need to select an automated test tool that suits your application and the testers (or developers) who will be creating your automated test scripts. There are lots and lots of tools out there, and choosing the right one can be difficult. Software testing tool review lists like [uTest's list of highest rated tools](#) can be a great resource on your search.

Once you've selected your tool and figured out who will be responsible for creating the automated tests, you'll need to make sure those team members have time to script. Obviously building out your automated suite won't happen overnight, but by resolving to create a few automated scripts each sprint or release, you'll gradually get there.

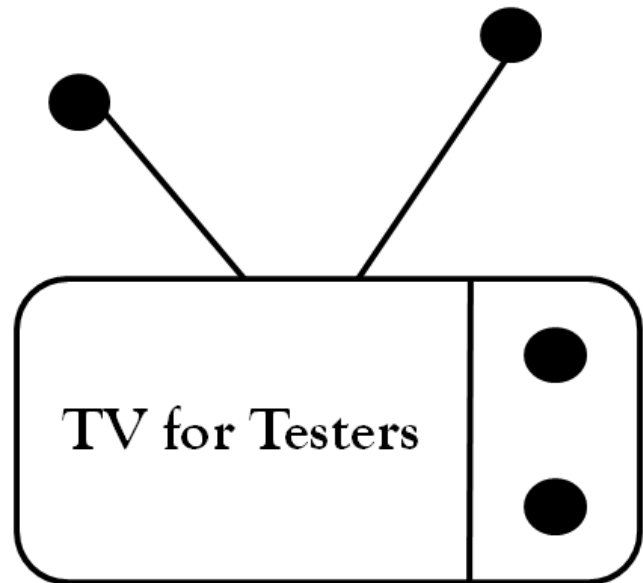
And once those automated scripts are running regularly, you'll be amazed at how automated testing can relieve some of your team members' burdens and how much more of your application you can test each cycle.

Stop thinking in terms of "or." Use both manual and automated testing to make sure your application is in tip-top shape.



Cullyn Thomson is a Project Manager at Tellurium and a regular contributor to the "Test Talk" blog. In her spare time, she enjoys cooking, knitting, traveling, and practicing yoga.

This article first appeared on PractiTest's "QA Intelligence" blog.



Tune in now!

WWW.TVFORTESTERS.COM

Master the Essentials of UI Test Automation: Chapter Three



- by Jim Holmes

REAL LIFE GUIDELINES THAT DELIVER RESULTS

You're reading the second post in a series that's intended to get you and your teams started on the path to success with your UI test automation projects.

Important note: After its completion, this series will be gathered up, updated/polished and published as an eBook. We'll also have a follow-on webinar to continue the discussion. Interested?

Register to receive the eBook

Check out the record replay of webinar conducted March 25th

Webinar

Master Test Automation

Dave Haeffner and Jim Holmes

Watch now

Chapter Three: People

As with everything else in software development, success comes through getting good people and giving them the tools they need to succeed. Lining up the right people for your successful project means finding ones with the right skills, or giving them time and support to grow those skills.

Getting or Growing the Right People

Please get this clear in your mind immediately: you can't take developers with no UI automation background and expect them to succeed at test automation without help. You can't take manual testers and expect them to succeed at test automation without help. You can't take non-testers, give them a fancy tool and expect them to succeed at test automation without a lot of help. Ensure you're setting your organization up for success by ensuring you're able to get or grow the right team members.

UI automation requires a specialized set of skills to be effective. Becoming adept at UI automation requires time and mindful practice. ("Mindful practice" is a term used for carefully chosen work/study/practice meant specifically to improve one's skills or knowledge. You can't just "go through the motions" and expect to improve. You need to dedicate yourself to improvement. Read more in Andy Hunt's [Pragmatic Thinking and Learning](#).)

Keep in mind that it's not enough to simply learn UI automation skills—just as you want developers who write great maintainable code, you also want people who understand how to write great maintainable automation scripts.

Finding great people in any corner of the software development domain is hard; finding people with great UI automation skills is much harder. It's a specialty that's underappreciated and not practiced widely enough. Make sure you have your time frame and budget expectations properly set if you're looking to expand your team by hiring.

What I've found more effective in my experience is to find current team members, or people elsewhere in my organization, and bring them to my team, then develop their skills. Poaching from other teams ("stealing" is such a harsh word!) can be politically tricky at times, so make sure you're not sawing off the branch you're standing on, metaphorically speaking.

Building UI Automation Skills

You'll need to work hard at building up your team's skills. This means you'll need both resources to learn from, plus a plan on how to learn.

Resources for Learning

Thankfully, there are a lot of great resources around to help you learn UI test automation. Better yet, many of the fundamental concepts are the same, regardless of the specific toolset you're working with, so you can look to industry experts rather than just tool experts.

Of course Telerik offers [training specific for Telerik Test Studio](#), but you can also look to some of these resources to help you build your domain-level knowledge of UI automation.

- Dave Haeffner's [Elemental Selenium newsletter](#).
- Dave's The Internet project on [GitHub](#) and [Heroku](#) is a set of common automation problems like forms-based authentication, asynchronous actions, drag and drop and so on. It's great for learning fundamentals.
- Marcel de Vries' Pluralsight course on [Coded UI](#) (subscription required).
- Richard Bradshaw is prolific both on [Twitter](#) and [his blog](#). His post on [handling setup and configuration for WebDriver](#) is full of great ideas.

Getting Your Team Learning

I've worked with many teams and individuals to help them up the learning curve, and I've talked with lots of other people who've been through the same process. All that experience and those conversations echo what should be common sense: directed, thoughtful practice is the best way to quickly get up to speed.

Remember: as you get to this point, it's critical that you've got support from stakeholders and your team. That support is key to getting through the learning process.

Here's an outline of a process that may be useful to you:

1. *Find a guide:* Look for someone with UI automation experience to help you and your team. Is there someone on a different team in your organization? See about getting them on your team, at least part-time. If not, hire an external guide to come in and help. Regardless of whether your guide is internal or external, this has to be a long-term commitment. You'll need more than a 2-day workshop; you'll need a relationship that will be highly involved for weeks, then ramp down over several months.
2. *Make a plan:* Lay out a roadmap for your team's learning experience. You'll need to think of things like general tool competency, creating backing APIs/helpers and, of course, creating the actual tests. Ensure all this work is in your backlog, work item tracker, Kanban board and so on. No training, skills building or actual automation work gets hidden!
3. *Pair up:* [Pairing](#) isn't just for developers. Pair less-adept team members with those who have more experience. You'll see benefits even if your team has no experience—putting two novices together, with frequent oversight and constant encouragement, can result in those two talking through problems and coming up with solid responses. (*NOTE:* This doesn't mean you can skip getting good guides for your team.)
4. *Focus on value:* You'll need to start with automation scripts that focus on small, "[low-hanging fruit](#)" features with which your team can be successful. That said, focus on scripts that will check honest business value for your stakeholders. Avoid wasting time on things like look and feel.

5. *Trumpet successes:* Make sure the team sees successes, especially as they're struggling to get up the learning curve. Figure out what metrics make sense to monitor for your team, and then get those up on a Big Visible Chart that everyone can see.
6. *Share knowledge:* Make sure that your team has the tools in place to share knowledge. Lessons learned, both positive and negative, are a huge help in the team's climb up the learning curve. Get some form of a knowledge base in place via Evernote, a wiki, or similar tools. Make sure it's easy to edit and search. Add "lunch and learn" brown bag sessions for the group to demonstrate concepts. Whatever you do, create the mindset that knowledge sharing isn't an option; it's expected.
7. *Constant feedback:* Software delivery benefits tremendously from constant feedback. Successful UI test automation projects, especially so. Prompt members to discuss automation during the teams' daily standup. Ensure your team retrospectives bring up good and bad points of your automation efforts. Most importantly, get stakeholder feedback on how they feel the effort is helping them: do the business owners feel they're getting better information to help them make informed decisions?

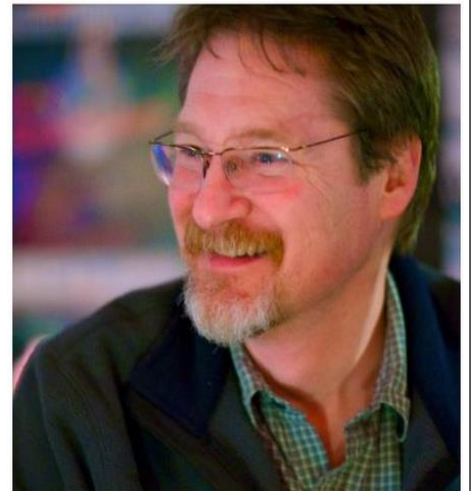
Your Team's Worth the Investment

Getting the right people lined up and empowered to succeed is crucial. Learning to master automation takes a long time, but the payoff is worth it: better value and higher-quality software delivered to your users.

Next in the series we'll talk about lining up the resources you'll need to be successful.

Jim Holmes

Jim is the owner/principal of Guidepost Systems. He has been in various corners of the IT world since joining the US Air Force in 1982. He's spent time in LAN/WAN and server management roles in addition to many years helping teams and customers deliver great systems. Jim has worked with organizations ranging from startups to Fortune 100 companies to improve their delivery processes and ship better value to their customers. Jim's been in many different environments but greatly prefers those adopting practices from Lean and Agile communities. When not at work you might find Jim in the kitchen with a glass of wine, playing Xbox, hiking with his family, or banished to the garage while trying to practice his guitar.





Sharing is caring! Don't be selfish 😊

[Share](#) this issue with your friends and colleagues!



However, in the next few years, our meetups and speaking sessions were attended by up to 80 people and most even paid to attend those sessions. I must say that AST Grants programme helped a lot arranging for initial sessions. Later, HKCS started assisting as well and that immensely helped me spreading the word about good testing.

From your experience around initiating and setting-up testing services, would you like to share some key learnings that might help the aspirants?

One thing that is absolutely important while setting up or initiating a new service is to understand the culture of the workplace. What I have noticed is that people come on board and try to implement the ideas that had worked for them previously. Those ideas may not be the right fit for the newer context. As consultants (because we consult our stakeholders) we must be ready not just for learning but also for unlearning. Once culture is understood and goals & objectives are clear, one can move forward with forming the team. The team development process remains same to what Bruce Tuckman proposed: Forming – Storming – Norming – Performing. While he added Adjourning to it later; I like to add Transforming instead because after a while complacency starts settling in and transformation becomes vital.

And when did your context driven awakening happen? We would like to know your experience with CDT community and your experience around rolling out CDT in organizations.

The CDT awakening happened when I first picked up ‘Testing Computer Software’ and ‘Lessons Learned in Software Testing’. By the initial years of the decade of 2000 I had read more or less all other prominent software engineering and testing books including Pressman’s Software Engineering which was a textbook. Testing books that I read included books by G. Myers, Ron Patton, Ed Kit, William Perry and Rex Black.

All those books had something to teach but I guess I was searching for a better answer to my questions. I was working for consulting firms and was talking the language of QA, CMM, ISO and Centres of Excellence etc. I was not happy with the way testing was being delivered. Both ‘Testing Computer Software’ and ‘Lessons Learned’ prompted me to learn more about context driven approach and as soon as I realized it was what I wanted, there was no looking back.

Now, about rolling out CDT in organizations, it was not at all possible to do working for consulting firms. Individually I continued doing so for the clients, educating them and telling them about it as that was the best way to go. Later, I applied some of it when I moved to Hong Kong. That was easier to do because I was managing a larger group. At my Australian employer, we have changed the methodology completely and we are very much a context-driven group of testers here. It is a feeling of Nirvana. :)

What according to you matters most in building intellectual testing culture at work place, especially in the large organizations?

Communication and education of stakeholders. Most people do not know much about what testers do, forget about how they do it. We still have people out there who think that anyone can test and sadly many of them are stakeholders for testing groups. These people need to be educated about what good testing is and what is needed to perform it. The process of education includes influencing the right people.

When I was in UK working for Boots Inc., we did roadshows to explain what testing team does and how it adds value. We got comments like, “I wasn’t aware that testers do that”, or “this brings a lot of benefits to my project” etc. So, it helps telling people what you do. Communication is one of the most important skills for testers.

What is the ideal form of influence according to you?

Rational Persuasion. It often works when you honestly tell people what you want and why. (The term Rational Persuasion was coined by Robert Dahl who was a political theorist.)

Another thing that helps influencing others is about understanding conflict. I wrote a blog post about this sometime ago. You can read it here: <http://www.dogmatictesting.com/2013/03/understanding-conflict.html>

And let me ask my favorite test management question. What metrics do you prefer to measure testing? What would be your advice to organizations who are still measuring testing by managing numbers?

Measuring testing by metrics makes no sense at all. Organizations (managers) use metrics as gun point to hold you accountable for things that have no meaning. In a recent release my team found over 100 bugs. What does it mean? It means nothing if you are not asking me probing questions like what was the context, what does it mean by finding 100 bugs, does it mean you found a lot of bugs for your context or it means that the number is reasonable etc. What is important is to consider the context. Counting those numbers mean nothing.

What people should be interested in is the software and not in how many test scripts are there or what is the number of bugs.

Your recent article titled “LinkedIn PDCA for Testers” has been well received by community. What do you think are most important skills that next generation tester must possess?

Communication, confidence, critical & lateral thinking, questioning ability and willingness to learn. Technical skills are nice-to-have and should complement the primary skills. Technical skills can be acquired.

Do you automate everything? Why not?

No, we don't automate everything. Why not? Because that is not possible. We do have automated checks available that do the basic stuff of checking, but testing is done by skilled testers.

Unfortunately, even today, decision about testing are made/influenced by stakeholders who don't have testing background. Have you ever gone through this? How should test managers deal with such situations so that good testing is not compromised?

I go through this often. One of the ways to deal with this kind of situation is to do risk profiling. Most stakeholders understand risks and many of them are risk averse. It is often easier to explain to them why compromising with good enough testing may create a risk for which they may have to respond. In one of my previous jobs, I created a presentation documenting major software failures of the year and how those failures cost dearly to the organizations on their reputation and business. It worked.

Your three skills that have helped you become what you are today, are....

Constant learning, Optimism and Integrity.

You have the reputation of being a leader in community as well as being a successful businessman. Which kind of people software testing community needs most today? What does community lose/gain when leaders become businessmen?

Software testing community needs people who are passionate about it. We also need people who are more interested in the advancement of the community rather than selling certificates. ASTs mission and purpose statement says it all.

I do not think that it matters whether community leaders become businessman or vice versa. What matters most is that there is no conflict of interest which we see in some of the areas beyond CDT community.

Looking back into those past 20 years, where do you think software testing has reached to? Are there still any things which are blocking the evolution of testing?

Recently testing reached to version 3.0. If I stand in the middle of Australia and say that loudly, a lot of testers and their managers, and recruiters and many more associated with testing will not understand it. However, anyone passionate about testing will understand it. This is where we are. It is indeed sad that many hiring managers still consider ISTQB a skill and consider it a must-have. I was saddened when few job-seekers told me that they are forced to mention ISTQB certification in their CVs to get it short-listed. This means that largely the recruitment processes are flawed. I direct people to Michael Bolton's post where he suggested about dodging the keyword screening by stating something like, "I do not have ISEB or ISTQB certification, and I would be pleased to explain why." It was a very clever idea.

In my current role, a lot of recruitment agencies approach me and offer their services. I am using this as an opportunity to educate them about CDT and about the need for skilled testing. I am very pleased that some of these recruiters have started screening candidates based on critical & lateral thinking questions rather than certification.

You have the experience of testing delivery and management across different domains and locations, ensuring that profit margins are achieved. Please share your secret formula?

It is like being an honest salesman (no, there is no contradiction here). You have to understand what your client wants and why. If you are able to put yourself in your client's shoes and understand their pain points, it is easier to help them solving their problems. See, honesty and integrity are still the best policies.

We know that there is special shared folder for Tea-time with Testers in your organization that testers regularly update and read. We would like to know your experience with TTwT and feedback for further development.

My experience with TTwT has been very positive and this is why I motivated my teams to read and save the magazine for future references. Your magazine is a constant source of knowledge through the variety of articles. We are content with the information presented by the magazine.

Happiness is....

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



Like our FACEBOOK page for more of such happiness

<https://www.facebook.com/TtimewidTesters>

Have technical skills, ideas, utilities or software tools to showcase to the world?

Write for next issue of **Software Tools Magazine**.

Your performance, our platform.

Let the show begin!



YEAR I ~ ISSUE II



SOFTWARE TOOLS MAGAZINE

STATE *of* TESTING



**Thank you for participating.
The report is coming soon!**

T ' Talks



T. Ashok exclusively on software testing

Beauty is only skin-deep

I am sure you have heard this idiom. So what does this mean? Searching for the meaning of this yields "External attractiveness has no relation to goodness or essential quality." (The Free Dictionary) This maxim was first stated by Sir Thomas Overbury in his poem "A Wife" (1613): "All the carnal beauty of my wife is but skin-deep."

In my conversations with testers, my observation has been that they seem to be focused on understanding the entity under test largely on the basis of the user interface. The testing seems to be driven by the UI metaphors and therefore the test cases don't reflect the depth sadly. They seem to largely dwell on the input variations and the states of UI metaphors and therefore seem superficial.

A beautiful interface can corrupt! It is imperative to go beyond the UI ('the skin') and not be infatuated by the external interface. After all good testing requires one to be curious of what is under the hood to ascertain if the behaviour does deliver the intended expectations.

What does it mean to look under the hood?

It is about understanding the data flows, the various components in these data flows, the interfaces that transport the data, state(s) of the system, conditions that transform the data, exceptions/errors that

may be generated, the system boundaries and other systems involved, and the various system resources needed.

Commence the dig by understanding what the visible and 'invisible' inputs are. The 'invisible' inputs are those data that are picked from file/database/shared memory/environment. Now dig in to understand the flow of data and their transformation. Identify various components through which the data flows. Understand where these various components are deployed, note that they may not be on the same machine. Having understood the various components, their location and the data flows, it is time to understand the interface details through which the data flows.

Do not forget that we need to understand 'over the hood' too. What is this?? Understanding who uses these, when and how much they use and what possibly they expect for a great experience.

Having understood the various components in the flow, we are ready to dig into the transformation logic. What are conditions that are used to transform the data? As we do this, we discover conditions that govern the functional behaviour. This allows us to model the behaviour and therefore logically come with up various (test) scenarios to evaluate the flows.

Having figured out the various components and the transformation conditions, we need to understand where these components are deployed and therefore the system boundaries. How does this help? To understand the various moving parts and where they are, to enable how shaky the system could be and therefore come with interesting scenarios to evaluate.

Hmmm... What next? Now that we know who the various components are, where they are, what the conditions of transformation are, it is time to determine the states of the various components ('when') and what they resources they need and what & how they can be affected by environment.

It is no more about being skin-deep, it is really about understanding the intended inner beauty enabling us to engineer the removal of blemishes.

Do not be blinded by the terms 'black or white', focus on using the inputs to construct the data flow and the transformation. The enquiry process to perform this is interesting & fulfilling, filling you with confidence as you 'picturise' the entire behavior.

Be it a brand new entity to test or an existing entity that I need to understand, I use this system of thinking to understand better. And in the process, come up with interesting questions!

So the next time you encounter a front end for an entity under test, go beyond the skin. After all a well-crafted feature should be hiding all the brutal complexity inside. Digging deeper is indeed a wonderful mental exercise!

So if I show you the interface for a search engine that has a simple text box to enter the search string, what will you do? Have fun digging!

Au revoir.



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



testing intelligence

- *its all about becoming an intelligent tester*



an exclusive series by **Joel Montvelisky**

Peripheral vision and peripheral testing

Back in high-school I was part of my school's basketball team, and I remember that one of the first lessons I got from my coach was about the advantages of peripheral vision.

He explained to me that peripheral vision in the game was incredibly helpful as it allowed you to:

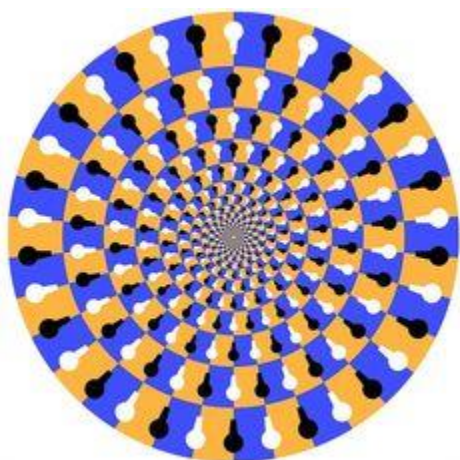
1. Look at one member of your team and pass the ball to another member, and so confuse the guys from other team while in defense.
2. It helped you to find the "open" guy in the team and pass him the ball at the right time (hopefully when he was alone under the rim).

3. And it also helped you to see if someone was approaching you in order to block you while you were trying to cover the guy dribbling the ball in front of you.

There were plenty of other things he mentioned, but the point was that he made us perform all sort of drills and practices (inside and outside of the basketball court) to help us develop our peripheral vision.

So much so that this is something that I still use today, while sometimes trying to handle my 3 kids alone, especially when we are in the mall or in the park and each of them decides to run on different direction to seek their own trills...

What is peripheral vision?



Almost everyone has some level of peripheral vision, even you have it!

Have you ever notice when you are walking in the street and all of a sudden someone is running in your direction approaching you from the side? The fact is that even if you were looking straight in front of you, you will still be able to “notice” that person, or car, or kid in the bicycle approaching you fast from the side of the street. Basically you caught their movement with “corner of your eye”, and that is peripheral vision.

I believe that this is one of those things we should have developed some 100K years ago as part of our evolution. Think about it, this might be the reason your great-great-great cave-dwelling grandfather did not get stomped by a big mammoth and his cousin did back in the day...

I think that some people have naturally better peripheral vision than others, but as I mentioned above, there are also drills and things you can do to enhance your peripheral vision and make it more accurate and refined.

Is there such a thing as peripheral testing?



Well, I guess if there wasn't such a thing I would not be writing this blog, right?

Peripheral testing is the process of finding bugs and issues in your product even if you are testing other areas and aspects of it (or even if you are testing another product altogether).

Just as in the case of peripheral vision, peripheral testing is something that most testers do every day – even without noticing it.

For example, when you are running a test flow in your product and you notice that the logo of your company on the top right section is cut. You were expecting to find a bug in the functional flow of the application and you were not looking or expecting to find this bug, but it caught your eye and so you found it.

The problem or the challenge with peripheral testing is that sometimes or some people might miss these bugs in the product because they were not looking for them. And I think that this is especially true for people who are only starting in testing, and also for those who tend to get “too focused” on their testing objective.

Can you develop your peripheral testing skills?

Absolutely!

I find that there are things that will help you to develop your peripheral testing skills, some of them are easier to practice than others.

1. Find your balance between focus and “dispersity”. By dispersity I mean the feeling when you are looking at all places at the same time (but not focusing on anything). By definition, in order to find the things that you are not looking for, you need to be open to find them when they appear, and this can only happen if you have room in your mind to detect them.
2. Make it a habit of “looking around” after every operation. This is something I try to teach some of the junior testers in my teams, put aside some time during your tests to look around your testing, this is very similar to the “focus/de-focus” technique but applied on a very low level. If you are running a functional flow that goes over a number of screen, make sure that you stop to look at the whole screen before you press the “next” button.
3. Learn to find your [testing zone](#). As trivial as it may sound, if you are completely concentrated on testing, and manage to leave all distractions out, you will be able to free some of your mental resources to find the stuff on the sides of your testing tasks.
4. Test twice, once for the functionality and the other one for the rest of the stuff. This is another exercise for junior testers, when you are running a testing operation run it twice. The first time to test the

functionality you were looking for, and the second one to see everything happening to the application while this functionality is happening.

5. Do more pair-testing. When 2 people are looking at the same thing while it happens, there are bigger chances that one of them will “catch” things happening on the side and alert the other to them so that both of them can research this.

6. Fight your urge to “robot test”. This is especially true for testers working on repetitive tasks, when you are running the same test for the 20th time, it is very easy to tune yourself off and let your mind wonder to any other place in the universe. In these cases the only way you will see a bug is if it blue-screens your machine more or less... Regardless of how many times you are tasked with running the same test you need to be on your toes and make sure to focus on the work at hand!

And finally, there are some things that come with time

Just like good whiskey, or like pain in your joints and back when you wake up, there are some things that will come with time...

As frustrating as this may sound, peripheral testing is one of those skills that testers develop with experience and time. The more you’ve tested, the more prone you will be to finding stuff that others simply pass over and miss completely.



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.

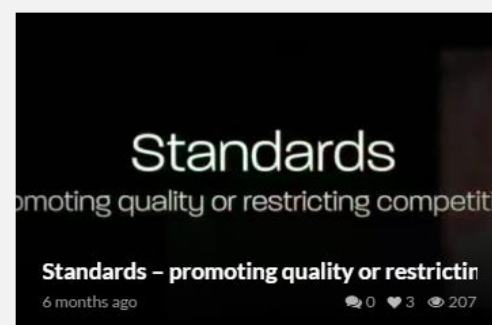
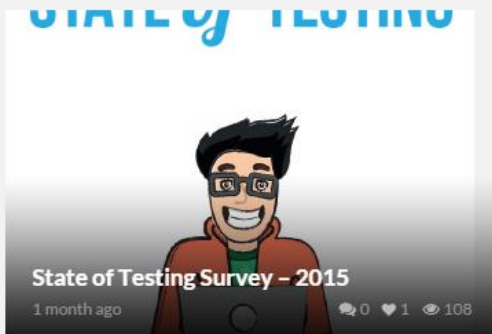
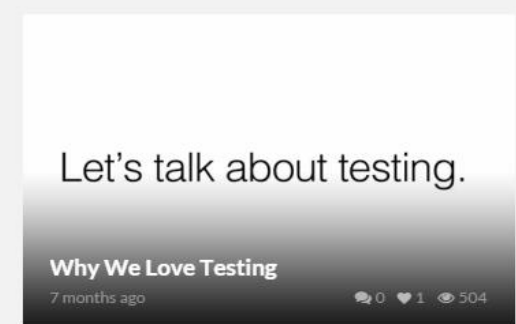
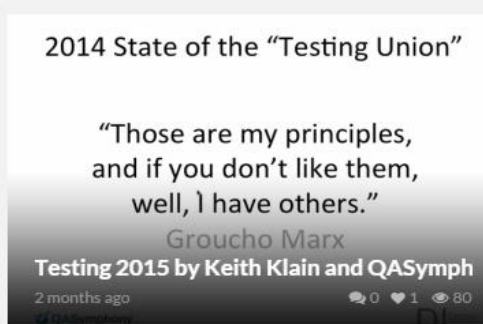
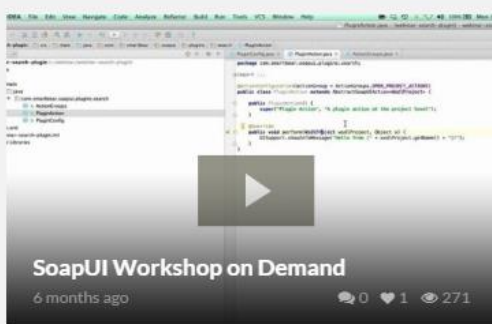
He also imparts short training and consulting sessions, and is one of the chief editors of ThinkTesting - a Hebrew Testing Magazine.

Joel publishes a blog under - <http://qablog.practitest.com> and regularly tweets as joelmonte


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

TV for Testers

Your one stop shop for all software testing videos



WWW.TVFORTESTERS.COM



www.talesoftesting.com

What does it take to produce monthly issues of a most read testing magazine?
What makes those interviews and articles a special choice of our editor?
Some stories are not often talked about...otherwise....! Visit to find out about
everything that makes you curious about **Tea-time with Testers!**

Advertise with us

Connect with the audience that MATTER!



Adverts help mostly when they are noticed by **decision makers** in the industry.

Along with thousands of awesome testers, Tea-time with Testers is read and contributed by Senior Test Managers, Delivery Heads, Programme Managers, Global Heads, CEOs, CTOs, Solution Architects and Test Consultants.

Want to know what people holding above positions have to say about us?

Well, hear directly from them.

And the **Good News** is...

Now we have some more awesome offerings at pretty affordable prices.

Contact us at sales@teatimewithtesters.com to know more.





Every Tester

who reads Tea-time with Testers,

**Recommends it to friends and
colleagues .**

What About You ?

in ne>xt issue

articles by -

Jerry Weinberg

T Ashok

Rahul Verma

Joel Montvelisky

...and others

our family

Founder & Editor:

Lalitkumar Bhamare (Pune, India)

Pratikkumar Patel (Mumbai, India)



Lalitkumar



Pratikkumar

Contribution and Guidance:

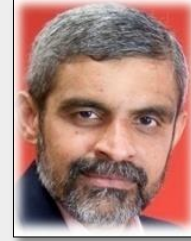
Jerry Weinberg (U.S.A.)

T Ashok (India)

Joel Montvelisky (Israel)



Jerry



T Ashok



Joel

Editorial | Magazine Design | Logo Design | Web Design:

Lalitkumar Bhamare

Cover page image – wallpaperup.com

Core Team:

Dr.Meeta Prakash (Bangalore, India)

Unmesh Gundecha (Pune,India)



Dr. Meeta Prakash



Unmesh Gundecha

Online Collaboration:

Shweta Daiv (Pune, India)



Shweta

Tech -Team:

Chris Philip (Mumbai, India)

Romil Gupta (Pune, India)

Kiran kumar (Mumbai, India)



Kiran Kumar



Chris



Romil

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

To get a **FREE** copy,
[Subscribe](#) to mailing list.

SUBSCRIBE

Join our community on

facebook.

Follow us on - [@TtimewidTesters](#)



www.teatimewithtesters.com

