

Tea-time with Testers

MARCH 2015 | YEAR 5 ISSUE 1

Jerry Weinberg

Where do Bugs Come From

Leah Stockley

Opportunities in Software Testing

Michael Larsen

Becoming a Little "Devious" for a Greater Good

Arslan Ali

Why choose Testing as a Challenge!

Keith Klain

Homework - Making the Best with What You Got

James Bach & Michael Bolton

Exploratory Testing 3.0

Martin Nilsson

Teaching New Testers about what it means to be a Tester

Parimala Hariprasad

What does it mean to be a Tester?

Varsha Shashtri

Sculpting NextGen Testing Minds

Over a Cup of Tea with Vu Lam

Early Test Collaboration for Successful Automation

Test Studio Release Webinar: April 28, 11 a.m. ET



What we'll cover

Collaboration isn't just a buzzword; it's central to a great project.

Join us for a 90 minute information-packed webcast featuring guest speaker Melinda-Carol Ballou, Program Director, ALM Research and Executive Strategies at IDC, along with Daniel Levy, Director of ALM, Telerik, who will discuss the importance of bringing testers and developers together for long-term success with automation.

We'll also demo the latest release of Telerik® Test Studio® test automation solution. Telerik Solutions Engineer Andy Wieland will introduce our new revolutionary "Intelligent Element Mapping" functionality, which supports early test collaboration between teams and speeds up time to market.

The Presenters:



Melinda-Carol Ballou
PROGRAM DIRECTOR,
IDC



Daniel Levy
DIRECTOR,
Telerik ALM Division



Andy Wieland
SOLUTIONS ENGINEER,
Telerik



Shravanthi Reddy
PRODUCT MARKETING MANAGER,
Telerik

[Register](#)



TeamQualityPro

SEE EVERYTHING



TAKE THE FIRST STEP IN SAYING:

- 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

To the Bright Future of Testing Field

A few months back I was approached by an HOD and Librarian of an engineering college in Coimbatore, India. And then again by few more colleges. They asked me for the permission to print some issues of Tea-time with Testers to be kept in their library for students and staff.

Well, that was great. It gives me an immense pleasure to learn that TTWT is not only read by testers within industry but college staff and students too find it promising. Then we thought, why not create a dedicated issue for them all together? We approached testers with coaching experience and experts in the field to contribute for this cause and the result is this issue.

All articles in this issue talk about the fundamental skills for testing, key concepts, practical advice for getting good at testing, choosing testing as a career and what matters when it comes to teaching testing to students/new testers. There are some articles on technical skills along with some inspiring stories from leaders in industry. I'm glad that this issue has come out great and hope that all level of our readers would like it.

This issue is our attempt to help college grads (and new testers) understand what meaningful testing is, how it is done on the job and what are other things that they must know, as opposed to their university syllabus for testing and before falling to prey to professional certification schemes in market or the training institutes that teach fake testing.

I don't expect to see things changed for better overnight but I hope that this issue would at-least help students to take informed decision about their future in testing.

For those who are interested, I regret to inform that next issue of Software Tools Magazine is delayed by some more days as out of 18 submissions, only 2 could pass our editorial tests. And publishing issues for sake of it is deliberately not our way. We are working on creating some quality content and we hope to launch it at the earliest.

That's all from my end for now. Stay tuned for some updates and announcements that is going to change the way you read Tea-time with Testers. Until then....

Sincerely Yours,



Lalitkumar Bhamare

editor@teatimewithtesters.com

@Lalitbhamare / @TtimewidTesters



QuickLook

Early Test
Collaboration for
Successful
Automation

Webinar:
April 28, 11 a.m. ET

Register



Editorial

What's making News?

Tea & Testing with Jerry Weinberg

Speaking Tester's Mind

Opportunities in Software Testing – 22

Becoming a Little "Devious" for a Greater Good – 26

Why choose Testing as a Challenge! - 29

Sculpting NextGen Testing Minds - 32

Letter to a Starting Tester - 73

In the School of Testing

Homework - Making the Best with What You Got – 39

Exploratory Testing 3.0 – 43

What does it mean to be a Tester? – 50

Teaching New Testers about what it means to be a Tester – 56

T' Talks

Like mysteries? Like to explore? Consider testing as a career- 71

Over a Cup of Tea with Vu Lam

Family de Tea-time with Testers



What's making News?

SmartBear Assumes Sponsorship of Swagger API Open Source Project

SmartBear Now Leads the Two Most Widely Adopted API Open Source Initiatives – Swagger and SoapUI

SOMERVILLE, Mass. – March 25, 2015 – SmartBear Software, the leading provider of open source API testing and development tools, has acquired the Swagger API open source project from Reverb Technologies. Swagger is the leading API description format used by developers in almost every modern programming language and deployment environment to design and deliver APIs that fuel IoT, microservices and mobile applications in the connected world. With this acquisition, SmartBear is now the company behind the two most widely adopted API open source initiatives, SoapUI and Swagger.

"Swagger has been the clear leader of the API description format discussion for several years – its ecosystem and passionate community is unsurpassed in the field," said Ole Lensmar, CTO at SmartBear. "We look forward to working with Tony Tam, Swagger's creator, to give Swagger the dedicated backing and support it needs for growth, primarily to ensure the open source project's evolution but also to ease its adoption into enterprise scenarios."

Swagger is a simple yet powerful representation of RESTful APIs. With Swagger, API developers can easily deliver interactive documentation, client SDKs and discoverable APIs. With its powerful code generation capabilities and open source tools, Swagger makes it easy for developers to go from design to implementation in a short amount of time. Swagger helps leading technology companies and enterprises like Microsoft, IBM, Apigee, Getty Images, Intuit, LivingSocial, McKesson, Morningstar and PayPal build the best possible services with RESTful APIs.

“Since first being released in 2011, Swagger has found broad adoption in start-ups, mid-size and enterprise companies alike—it has grown far beyond what we had envisioned,” said Tony Tam, CEO at Reverb. “Now it’s time to take Swagger to the next level and we have chosen to partner with SmartBear because they have the API expertise and proven commitment to open source with products like SoapUI. With SmartBear, Swagger will reach more developers, products and services, and make an even bigger impact on the API world.”

SmartBear is committed to keeping the Swagger specification and code open and driven by the community, and encourages contributions through evangelism, documentation and tooling. The company is engaging industry leaders to create an open governance model that supports the evolution of the Swagger specification in a vendor-neutral and collaborative manner.

As part of its commitment to Swagger, SmartBear will be investing in development to evolve the specification and toolset, as well as providing commercial support offerings for enterprises using Swagger. The company will also be developing and providing resources to help developers adopt and use Swagger and the Swagger tools.

For more information, contact us at apiteam@swagger.io. For information on Swagger, visit: <http://swagger.io/> and follow us on Twitter [@swaggerapi](https://twitter.com/swaggerapi).

About SmartBear Software

As the leader in software quality tools for the connected world, SmartBear supports more than two million software professionals and over 25,000 organizations in 90 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.

Are you interested in publishing the news about your own firm, tools, community and conferences in **Tea-time with Testers?**

Then write to us at:

sales@teatimewithtesters.com

with “**News Enquiry**” in your subject line.





A million dollar smile ?

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

*Adverts starting from \$100 USD | *Conditions Apply

Meet Mr. Vu Lam, the CEO and Co-Founder of QASymphony. He has significant and diverse software industry experience in the U.S. and international markets, including Asia and Europe.

An entrepreneur at heart, Vu has started and grown multiple technology ventures, including Paragon Solutions, a U.S. based technology consulting company with international development centers in Bangalore, India, and Ho Chi Minh City, Vietnam. Following Paragon's acquisition by NASDAQ traded FCG, he led the growth of the organization from 400 to 1300 resources. In 2011, Vu co-founded QASymphony.

Trusted for his leadership and deep industry knowledge, Vu is a board member for several technology startups. Vu received a Master of Science in Electrical Engineering from Purdue University and a Bachelor of Science from University of Illinois at Urbana-Champaign.

It's always a pleasure to discuss testing with Mr. Vu. Read this interview to find out what one of the creator of QASymphony thinks of testing, his career journey and much more.

- Lalitkumar Bhamare

Over a Cup of Tea with Vu Lam



It's an honour speaking with you today, Mr. Vu Lam. Would you like tell us about your career journey? And about your testing career?

Thank you! My journey started at the University of Illinois where I received my electrical engineering degree, after which I was hired at Bell Labs and was sent by them to pursue my masters in electrical in engineering at Purdue. It was during my time at Bell Labs that I was first exposed to testing. I worked with a group of developers in which half of our job was to write code and half was to test code of our teammates.

This fostered a competitive culture between us, where every developer was saying, "no one's going to find a defect in my code!" So really everyone became a better developer because they were forced to think like testers. I was trained there to be a tester and to have a testing mindset. I had to have my developer hat on and my testing hat on at the same time. Plus this process gave us all a better appreciation for what testers do and what their job really brings to the table.

Please tell us about your Context Driven Testing experience and journey...

Two and half years ago at QASymphony, we released a product called qTrace and we received a call from Barclays saying that one of the consultants we hired, Michael Bolton, introduced them to the product. Barclays thought it was exactly what they needed in order to test the way they wanted to test -- with a few minor exceptions. Barclays asked if we'd be willing to make some changes to adapt to their needs, and of course -- as a startup -- we jumped at the opportunity.

Michael Bolton, the consultant that introduced Barclays to qTrace, happens to be a big proponent of Context Driven Testing. So over the next 6 months we spent a lot of time studying Context Driven Testing and I became a believer in the philosophy. Since we're steered our product toward this school of thought to make sure that we became a toolset that supports this way of testing.

One of your companies, QASymphony is very famous among thinking testers worldwide. Please tell us story behind starting QASymphony...

After I sold my previous company, I started KMS in 2009 with a lot of "lessons learned" in mind. A couple of years into it, we decided that we should be building products ourselves. We saw the opportunity in the testing space with the move from Waterfall to Agile and from hosted to cloud -- no other companies were addressing this problem of how to test software now that development was happening in a matter of days and weeks rather than months and years. So we started QASymphony to fill that gap and provide tools for testers in agile environments. And now the company has been around for 4+ years, I think we've built something really great for the testing space.

QASymphony's product qTrace (now qTest eXplorer) is being considered as revolutionary and exemplary addition to testers' tool-kit. What inspired you to come up with such product? What was the key idea?

The inspiration for the tool actually came from the big need for training within the SAP world around the 2000 timeframe. Companies were having to train hundreds of thousands of people -- for example, around this time I was involved with a project with GM Europe where they had to train around 130,000 people on SAP, which is a very complex software. So we created a product that recorded an expert user going through SAP and generated descriptions of their path, and generated simulations allowing the user to read about what they need to do and then try what they need to do. Since that time, our product has been the number one tool for training SAP users. And with that technology in mind, we began to explore the idea of using the tool for testers. Once we got the call from Barclays, qTrace really began to move into the direction of the testing product that it is today.

... CONTINUED ON [PAGE 66](#)

Tea & Testing



with

Jerry Weinberg

Where Do Bugs Come From

When I first published this list in 1981, it was widely distributed by fans. Then for a while, it disappeared. Every year, I received a request or three for a copy, but by 2000, I'd lost the original. This month, when closing up our cabin in the woods, it turned up in the back of a file cabinet. Inasmuch as I've been writing recently about testing (as in **Perfect Software**), and as it seems still up-to-date and relevant, I decided to republish it here for the grandchildren of the original readers.

Not so long ago, it wasn't considered in good taste to speak of errors in computer systems, but fashions change. Today articles and books on software errors are out-numbered only by those on sex, cooking, and astrology. But fashion still rules. Everybody talks about debugging—how to remove errors—but it's still of questionable taste to speak of how the bugs get there in the first place. In many ways, the word "debugging" has injured our profession.

"Bug" sounds like something that crawled in under the woodwork or flew through a momentary opening of the screen door. Misled by this terminology, people have shied away from the concept of software errors as things people do, and which, therefore, people might learn not to do.

In this article, I'd like to shift the focus from debugging—the removal of bugs—to various forms of putting them in the software to begin with. For ease of reference, I'll simply list the various types of bugging alphabetically.

Be-bugging is the intentional putting of bugs in software for purposes of measuring the effectiveness of debugging efforts. By counting the percentage of intentional bugs that were found in testing, we get an estimate of the number of unintentional bugs that might still be remaining. Hopefully, we remember to remove all the be-bugs when we're finished with our measurements.

Fee-bugging is the insertion of bugs by experts you've paid high fees to work on your system. Contract programmers are very skilled at fee-bugging, especially if they're treated like day laborers on a worm farm.

Gee-bugging is grafting of bugs into a program as part of a piece of "gee-whiz" coding—fancy frills that are there to impress the maintenance programmers rather than meet the specifications.

Knee-bugging is thrusting bugs into a program when you're in too much of a hurry to bend your knee and sit down to think about what you're doing. We have a motto—"Never debug standing up." We could well extend that motto to bugging: "Never bug standing up."

Me-bugging may involve numerous forms of bug installation," for it refers to the method by which they are protected from assault. The programmer regards any attempt to question the correctness of the code as an assault on his or her value as a human being. Those who practice "egoless programming" are seldom guilty of me-bugging.

Pea-bugging can be understood by reference to the story of The Princess and the Pea. The princess showed her true royalty by being able to feel a pea through a hundred mattresses, illustrating that no bug is too small to be noticed by computers or other royalty. The pea-bugger, however, pops in little bugs with the thought, "Oh nobody will ever notice this tiny glitch."

Pee-bugging is the rushing in of bugs when the programmer is impatient to attend to other matters, not necessarily the call of nature. The pee-bugger is the one who is always heard to contribute: "Let's just get this part coded up so we can get on to more important matters."

Plea-bugging is the legalistic method of forcing bugs into software. The plea-bugger can argue anyone out of any negative feeling for a piece of code, and is especially dangerous when walking through code, leading the committee around by its collective nose.

Pre-bugging is the art of insinuating bugs into programs before any code is written, as in the specification or design stages. We often hear the pre-bugger saying, "Oh, that's clear enough for the coders. Any moron could understand what we mean."

Re-bugging is the practice of re-introducing bugs that were already removed, or failing to remove bugs that were found and supposedly removed. Re-bugging is especially prevalent in on-line work, and some on-line programmers have been known to collapse from malnutrition when caught in an endless loop of debugging and rebugging a related pair of bugs.

Sea-bugging is named for the state of mind in which it is done—"all at sea." The typical sea-bug splashes in when everyone is making waves and shouting, "We've got to do something!" That something is invariably a bug—unless it's two bugs.

See-bugging is the implantation of bugs that "everyone can see" are correct. See-bugging is done in a state of mass hypnosis, often when too many susceptible minds are pooled on the same team or project. This unhealthy state prevails when management values harmony over quality, thus eliminating anyone who might make waves. Of course, too many wave-makers leads to sea-bugging, so programming teams have to be constituted as a compromise between harmony and healthy discord.

S/he-bugging is done all the time, though nobody likes to talk about it. S/he-bugs have a way of infusing your code when your mind is on sex, or similar topics. Because sex is a topic unique unto itself, all s/he bugs originate by what might be referred to as sexual reproduction. That's why this is such a populous class of bugs.

Tea-bugging is the introduction of bugs when problems are solved during tea and coffee break conversations and then not checked before passing directly into the code.

The-bugging (pronounced thee-bugging) is crowding multiple bugs into a program under the "one-program-one bug" fallacy. The addicted the-bugger can invariably be heard ejaculating: "I've just found the bug in my program."

We-bugging is the ordination of bugs by majority rule. When someone doubts the efficacy of a piece of code, as in a review, the majority votes down this disturbing element, for how can three out of five programmers be wrong? Thus are born we-bugs.

Whee-bugging is a symptom of boredom, and frequently arises when Agile programming and other bug-prevention schemes have become well-established. Programmers reminisce about "the good old days," when programmers proved their machismo by writing code in which nobody could find bugs. One thing leads to another, and eventually someone says, "Let's do something exciting in this piece of code—wheeeeeel"

Ye-bugging is a mysterious process by which bugs appear in code touched by too many hands. Nobody knows much about ye-bugging because every time you ask how some bug got into the code, every programmer claims somebody else did it.

Z-bugging is the programmer's abbreviation for zap-bugging. The zap-bugger is in such a hurry that it's unthinkable to use three letters when one will do. Similarly, it's unthinkable to submit to any time-wasting controls on changes to working code, so the Z-bugger rams bugs into the operating system with a special program called ZAP, or SUPERZAP. Although Z-bugs are comparatively rare, they are frequently seen because they always affect large numbers of people. There's a saying among programmers, perhaps sarcastic, "All the world loves a Z-bugger."

So there they are, in all their glory, nineteen ways that people put errors in programs. There are others. I've heard rumors of tree-bugs, spree-bugs, agree-bugs, tee-hee-hee-bugs and fiddle-dee-dee bugs. I'm tracking down reports of the pernicious oh-promise-me-bug. Entymologists estimate that only one-tenth of all the world's insect species have been discovered and classified, so we can look forward to increasing this bugging list as our research continues. Too bad, for we seem to have enough already.

Note added in 2015: Yes, we had enough, but many have been added. If you know of a bug we didn't list here, I invite you to let me know about it.

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](#) .

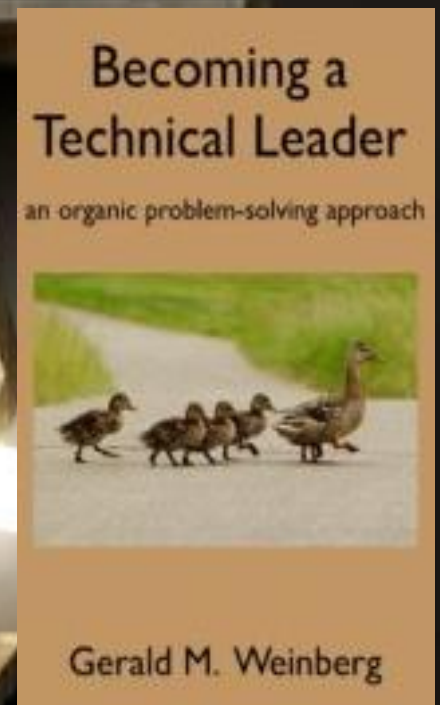
Gerald can be reached at hardpretzel@earthlink.net or on twitter @JerryWeinberg

Becoming a Technical Leader is a personalized guide to developing the qualities that make a successful leader. It identifies which leadership skills are most effective in a technical environment and why technical people have characteristic trouble in making the transition to a leadership role. For anyone who is a leader, hopes to be one, or would like to avoid being one.

This is an excellent book for anyone who is a leader, who wants to be a leader, or who thinks only people with 'leader' or 'manager' in their title are leaders.

Its sample can be [read online](#).

Know more about Jerry's writing on software on [his website](#).



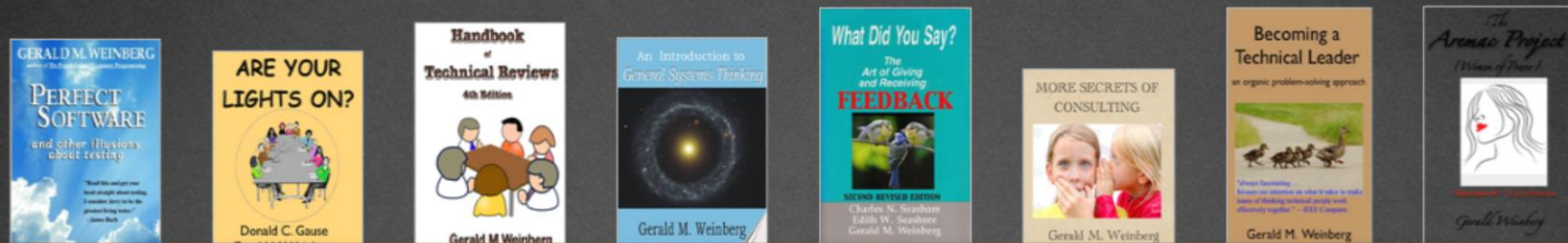
TTWT Rating: ★★★★★

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.

What Windows 10 Means for Testers

- by **Nikhil Kaul**

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.



A green pendulum bob hangs from a thin wire, positioned centrally over a circular pattern drawn in the sand. The sand is light-colored and textured, with the pattern consisting of concentric, wavy lines. The entire scene is framed by a dark blue border.

Speaking Tester's Mind

- straight from the author's desk

Opportunities in Software Testing



- by Leah Stockley

It was January 1999, the Y2K deadline was fast approaching and many companies were desperately hiring people to test all of their software and ensure the world didn't implode at 00:00 on Jan 1st 2000. In a small town in southern England, an advert appeared in a local newspaper (that's how people found jobs back in those days!), it simply said:

"Logical thinker wanted, no experience necessary!"

I answered the advert, undertook some psychometric tests and found out the role was for a Software Tester at a local bank. That was my entry into the world of software testing... 16 years ago it was nearly impossible to hire experienced testers, so the company I worked for invested in hiring people with the right mind-set and trained them to do the job. It was a lucky entry into what is undoubtedly the right career for my inquisitive mind!

Right now, in 2015, software testing is a recognised career path. People with a focused software-testing career behind them are reaching Managing Director level in multinational companies. Yet the majority of universities around the world are still not teaching it! I was recently invited to guest lecture at the National University of Singapore School of Computing. There, a professor has recognised this issue and launched a Software Testing module but can only accommodate 60 of the 2000 SoC students each year. But even this course is aimed at teaching developers the importance of testing their code (a valuable skill) rather than skills for the thousands of graduates who will become career software testers.

So does this reflect the real world situation?

A senior manager from a large systems integrator recently told me that on any given day they have more than 1,000 software testing roles unfilled! That's just one company... another similar company has more than 20,000 testers in one city alone! The fast changes in technology, the vast array of devices and numerous ways of accessing information is making Testing ever more necessary and complex. There clearly are numerous vacancies in the software-testing field, and I am hopeful the industry is learning that it can't just fill these with unskilled people... it needs more 'logical thinkers' and other aspects that make up the right mind-set to be an excellent tester.

But why be a tester when I could be a developer?

Let's get the facts out there early... There is generally more money to be earned and perhaps more glory to be had becoming a developer. In many companies the developer is 'king' and testers are seen as less important (not true of all companies but that mentality could exist amongst individuals anywhere). Developers are seen as the creative ones, making the product work. So if that is true, why would anyone consider testing? Ultimately it's down to you and your mind-set. Personally I know Development would have been the wrong career for me; my coding hobby is satisfied by using automation tools to aid testing, but more than that, I have found that working out the best way to test something from the hundreds of possible approaches is equally, if not more creative than writing code to satisfy a requirement. Testing gives me more opportunity to interact with people, speak to users and developers, and to gain understanding of the business or users I'm testing for. And yes, perhaps developers get more glory for creating the product (I've never yet seen the headline "tester saves the day" in a newspaper) but if you are the person that finds the bug and stops the next NASA Climate Orbiter from disintegrating as it enters Mars' atmosphere, saving the loss of \$327m, or you are the one who catches the defect that would have seen your company fined millions of dollars by the banking regulators, you will certainly become the hero or heroine within your organisation! But as I said, it's really about you and your mindset...

Who should consider testing as a career?

So we've established there are numerous opportunities to get into testing but does that mean you should rush into it? Please no!!! I am evidence that testing can bring a varied and rewarding career but like any career, it won't suit everyone.

So how can you determine whether it's right for you? Here are some questions to help you decide...

- Do you like to be given clear instructions before starting on a task?
- Do you prefer to work alone?
- Are you uncomfortable with change?

If you answered yes to these I'd advise you NOT to seek a career in testing. Testers constantly deal with ambiguity (in fact for some of us, seeking the facts is half the fun of the job). You have to like working with other people; we must interact with everyone on the project to gain the information we need to test well. As for dealing with change... without it there would be no need for testers, so we must learn to embrace it and be flexible enough to respond to it.

Here are some more questions to help you decide if a Testing career might be for you...

- Do you like variety in your job?
- Do you like to take things apart to get a better understanding of how they work?
- Are you comfortable speaking up, raising your concerns and providing logical explanations for them?
- Are you curious, always asking questions and eager to know more?

If the answer to these questions is yes then Testing could be the perfect career for you! Of course you also need attention to detail, a logical mind and the ability to communicate clearly with both technical and non-technical people.... And a whole host of other skills that I don't have room to list!

But is there a future in testing...?

There have been a few debates lately about whether testing as a profession even has a future... Traditional testing (commonly linked with the Waterfall methodology) is a very risk-averse, change-averse process. Significant effort goes in to attempting to create 'perfect' documentation (requirements specifications, test cases) and then trying to prevent further changes on the project, so the documents remain valid! Some testers will tell you this is essential or they cannot test. However many companies realise that this 'traditional' approach slows projects down. The test team can often be the 'police' for this process so companies see testers as preventing timely delivery of the software the business really needs. No wonder these testers are unpopular with the developers and project team.

Many companies are now attempting to adopt Agile because they accept in this fast paced world, where companies like Facebook and PayPal make multiple releases in a single day, the customers are demanding more and the global competition for market share is fierce. For these companies it's no longer acceptable for testing to take months to complete, during which time no other improvements can be made to the application!

A student asked me the other day why I was talking to them about testing as a career when most companies now encourage developers to do the testing. I actually think this is a misconception about Agile. Talking with Agile experts like Pete Deemer of GoodAgile reconfirms this. Companies adopting Agile need multi-skilled team members... testing is absolutely one of those skills.

Testers in these environments have to accept that we constantly work to hit a moving target. Those who don't accept this may soon be obsolete. However, testers who build excellent analysis & questioning skills,

exploratory testing skills and understand coding and how to use automation tools in a tactical way (e.g. using cheap tools to quickly generate millions of lines of test data, to free up extra time for actual testing) will absolutely have a future and earn themselves jobs in the most dynamic & interesting of companies.

Other aspects of testing in which there are definitely not enough experts are Performance and Security testing. Both of these are becoming ever more critical as our entire lives become dependent on technology and all of our information hangs in a cloud! A great place to specialise, both roles require good technical skills and both generally pay more than a manual testing role would.

How to find out more?

If you are thinking testing could be the career for you, you may be wondering how can you get started? Reading **Tea-time with Testers** is obviously an excellent start! It can help you find the names of industry experts, by following them online; you can start to learn much more about testing. But simply reading is not enough. Look in your local area and see if there is a Tester meet up, for a chance to meet other testers face to face.

Or why not get your hands dirty and try out some actual testing. You can join a crowd-sourced testing company like uTest, and take some assignments to start practicing your testing skills. You can even spend your free time testing sites like Facebook, who offer an average \$500 for every valid bug raised by a member of the public. Demonstrating that you have used your initiative and already begun your testing career to a potential employer will help you stand out from the crowd! And with hundreds of thousands of IT specialists graduating yearly, that is a skill you definitely need to perfect, whatever career you ultimately choose.



Leah has more than 16 years experience in Software Testing for consultancies, banks and software houses across multiple industries. In the last 4 years she has become passionate about inspiring and coaching testers to be innovative, empowered and to continually improve the art of testing. Currently an independent consultant, Leah works with clients, enabling them to gain the most from their test teams by defining & improving their Test Strategies, and applying new and efficient approaches such as Context Driven Testing & Exploratory Testing.

When not at work, Leah shares her experiences on www.inspiredtester.com. Leah is also the co-founder of the Singapore Testing Forum.org, a free networking event bringing Singapore based software testers together to share their experiences and challenges.

Becoming a Little “Devious” for a Greater Good

- by Michael Larsen



One of the most important tools a software tester can learn to use rests within themselves. It is their own mind. It is the ability to ask a simple question... “What if?” This might seem ridiculously straightforward, and perhaps condescending. I assure you it is anything but that. The challenge we face today, as we develop an ever more interconnected world, where so much information is available with a search or a collection of links, is that we lose a bit of our own memories in that process. We start to look for things that work, things that are “good enough”, or answer the immediate need. It’s so easy to fall into the comfortable trap of doing the bare basics, claim we have performed due diligence, push products out the door and call it a day. Of course, later on, we see problems that spring up. When we are asked why we didn’t see that particular issue, we have a true, but unsatisfying answer. We didn’t see that issue because we didn’t consider using the product in that way

As this issue is dedicated to those about to leave school and embark on new journeys, I want to consider thoughts and ideas I could share with those just getting started in the workplace. Specifically, I want to talk to those who have decided to make software testing an area of focus. If you have chosen to make software testing a career, you already have a tremendous advantage in that very act. You have chosen it, rather than let it choose you. Please note, many of the best software testers I’ve known through my career just “fell into the job”. I am also one of those who fell into it. In the process, I had to learn a lot of things that were never taught in schools. One of the most valuable skills I was ever introduced to came from a former co-worker early in my testing life. That skill was the power of “being devious”.

Software testing isn’t clean and elegant, much as we wish it could or should be. Learning fundamental rules? Sure, you can learn some of that from books. Learning to be devious? There’s only one place to develop that, and that’s through experience. For some, that ability does not come naturally. It requires that a person develop cynicism. It requires that they be able to think of ways that a product could be used for unsavory purposes. It requires a person to consider the evil that people might do, and then give themselves the permission to be “evil” themselves. For many, that last part is the hardest. Most people

are, generally, not looking for ways to break things, or to cause problems for others. We manage to do plenty of that on our own unintentionally. Most people want to do what they need to do to get along with others, cover the necessities of life, and allow for a few wants and pleasures to be experienced as well. In this quest, it's very easy to get caught up in the day to day essentials, and start to lose focus on the immense variability of things we take for granted every day.

A favorite quote of mine is attributed to the guitarist Leslie West. He once said "I can't teach you how to play guitar. I can show you how to play guitar. What I can do, though, is teach you how to teach yourself". Likewise, I can't teach you how to be devious, but I can show you things that helped me think deviously. When I create a program, very often my focus is on getting something to work. I think about the steps I need to accomplish, and I see ways in which I can make that happen, and often, once I have achieved my goal, I am done with it. There is a purpose, there is a framework to meet it, and then there is a syntax that makes those steps happen. As a software tester, my goal is to look at the same program and now think "how can I exploit and damage this program, to the point that I can render it either unusable or compromise it so that I can do something else with it?" How flexible is this program? What if I don't know all the options... can I discover them? If I were to run it with root permissions, could I get access to materials I shouldn't? Is there sensitive information I am accessing? If so, are there protections to prevent others from seeing it? Can I compromise those protections?

Financial software goes through lots of security checks and other methods to make sure that users don't do things that could cause them to lose money or credit things incorrectly. That's an easy place to look because, well, there's a chance for "free money". That's a good starting point, but what are other areas that would be fruitful to the unscrupulous? Why would you want to see if you could use a program to get root access? What might be there that could be valuable? Do you want to gather personal data? Do you want to manipulate figures to make yourself look better? Do you want to try to crack into the database so you can list yourself as having the top score in a game? Think of your motivations, even if you personally would never dream of doing such things in your personal life. Why might someone want to compromise your program? Is there a potential benefit to doing so? Could it enrich you, or could you learn something that would give you an advantage? This may sound silly, but just by asking those questions, you start to look at your program or application differently, and by looking at it differently, you start to consider different avenues for testing that would not come to mind otherwise.

At this point, it might be tempting to think that there are a variety of tools that a tester can use to help accomplish these tasks. The answer to that question is, of course, yes, but I think there is too great a tendency to jump to tools to do the work we want to do. Tools can be great helpers, but they will be much less effective than if we have spent some time understanding what we want to use the tools to do in the first place. A phrase I encourage any new tester to embrace is "forget the tool, start with the problem". Rather than start the conversation with what tools I will use, the first and foremost questions I need to consider are "where are the problems in this application likely to be? What would happen if someone who is less than honest or scrupulous were to decide to use this program? What would they do with it? Could they use it to do bad things? More importantly, can I think of how they might do so?"

For those who are just starting out in software testing, there is a great deal of variety and options you can explore. A lot of emphasis is placed on confirming that software works. This in and of itself is important, but we are not there to confirm that something works. Instead, we are asked to discover where a product is broken, and from there, communicate to others how what is broken could be a problem. The more willing we are to consider ways a product may be used for less than legitimate means, the more likely we are to find areas in which that broken nature can and will be found.



Michael Larsen is a Senior Quality Assurance Engineer with Socialtext in Palo Alto, California, USA. Over the past two decades, he has been involved in software testing for a range of products and industries, including network routers & switches, virtual machines, capacitance touch devices, video games, and client/server, distributed database & web applications.

Michael is a Black Belt in the Miagi-Do School of Software Testing, President of the Association for Software Testing (AST), a lead instructor of the Black Box Software Testing courses through AST, and a founder and facilitator of the Americas chapter of Weekend Testing.

Michael writes the TESTHEAD blog (<http://mkltesthead.com>) and can be found on Twitter at @mkltesthead. A list of books, articles, papers, and presentations can be seen at <http://www.linkedin.com/in/mkltesthead>.

Need more from Michael? Check this out!

IN THROUGH THE SIDE DOOR

A video talk by Michael Larsen



<http://tvfortesters.com/video/in-through-the-sidedoor-michael-larsen/>

Why choose Testing as a Challenge!

- by Arslan Ali

Well, you don't need to, if you don't know how to take up to a challenge – otherwise you are most welcome to it.

There was a question long time back on an on-line professional forum about having lack of specialized resources for the variety of tasks that are required by the ICT industry. Also, in the same discussion some points were raised in concern about the academia and industry that they are too focused on their product technical demands and not on the ideas and talents they can exploit for creation.

The fact is simple; every business in the world of any kind needs human resources to run it and flourish it, and also to perish it. We all read about this daily in the papers and tabloids that how businesses have grown from just a garage to a multimillion dollar company and how it has happened vice versa to a rocket boosting company which now resides on its knees.

The origin of human resources for a professional world context is basically from the academia side, and this is where the young brains are trained and flourished to become what they become in future.

For the ICT Industry same thing happens. The computer industry is about Information and technological solutions, how to build them, their complex architecture, databases, network and communication technologies and then What?

Exactly!

The academia is all focused on training and developing individuals to become "Architects and Developers" – they are so busy in doing so with their syllabus and courses, and lectures and projects that now they

don't have time to induce any other science as an specialty to teach students. On the other hand the industry is divulged into creating, marketing and then maintaining their products, and then keeping up to the technological demands from within or from their customers. This eventually creates a mindset that computer science is about Engineering and Development of solutions and it is about technology which helps in building these solutions. This concept actually puts the client and the actual product somewhere in the background of the picture – it seemed okay if you have a short list of customers and small scale apps, but it is a disaster if you are on a growing path, and carrying this belief like a magical cloak on your shoulders.

Software testing and quality assurance puts this framework of thought and mindset to a different perspective. It opens up a new door which is more towards the customer and product characteristic as a usable artifact. It addresses the concept of having a set of "Electronic Parts" distributed here and there, and once they are put together they become a "Mobile Phone", which actually brings "Value" to its users. Software testing also addresses the "Feeling and Psychology" of the product users, and does not look into the logic and structural errors, but the effect of these "Bugs" on a customer, not only the bugs, but how the product actually makes a user feel - happy, sad, frustrated, motivated, embarrassed, angry, or energetic... etc.

To become a software tester, where you have been premeditated with the computer science or any other business science to its core, you need to understand this equivalence and then have firm believe on it. You need to place yourself as a "Human" element, with knowledge, and decision power, which can analyze, coordinate, and use appropriate tools and skills in order to test any system under any time scale.

We need to understand the fact that Software Testing is not about computer science, but having the knowledge and experience of the latter, which actually helps. That is why we see so many implausible testers doing testing who are actually business and management sciences graduates.

The academia and industry now should realize that having someone to manufacture and create a product as resources is not adequate to deliver a viable product. Time has evolved and with it, the products and their users. A mobile company has no idea that who will use their high tech marketed product; would it be a business professional, a housewife, a teenager, a senior citizen, a mobile application developer, or a child?

We need Software Testers with specialized roles to cater for the complex development and product architectures. For that we need to induce this ideology from the academia level. Where the students can realize that besides learning how to develop systems, we also should know how to test them and provide working products to the clients. This is the reason why Test Automation is making more noise over the "Manual Testers" – because once you have only development population coming out of the academia, you will give them what developers know how to do, and in this case the scripting.

This whole cycle of acquiring and managing resources have put us into a very challenging position, as we have finally realized that without that "Context Driven" approach to test the product, and explore it with intent to finding important bugs, we cannot achieve the level of "Perceived Quality". The challenge is to shift this realization to the development and engineering side. The challenge is to create roles of testers on specialized areas, where they can assist the development and engineering process.

If we just chatter about "automating" everything in test, and distinguish this as the only way out, then we are actually placing ourselves in a blindfolded formation. People tend to lose focus that way from the complete framework of testing, project management, product, and quality perception, and brings it down to a mere testing technique. Small talk right, but that is what we face with an immense magnitude.

This is the core reason, why testing as a career, developing a mindset for testing and even teaching about it matters.

If you think that this is really a challenge to accept then at least you have taken a step forward – let's join hands and bring along a change which can help out students, enthusiastic professionals and testers around the world to recognize themselves and their talents.



Arslan Ali is a Software Testing and Training professional; he serves his passion at the OuttaBox (www.outtabox.co) as a Training Consultant for various software testing workshops, and also works as a Senior Consultant Information Solutions at Sidat Hyder Morshed Associates – a renowned software solution provided in Pakistan. Arslan has been around ICT industry past 15 years and have diverse experience in Software Development, Quality Assurance and Business Process implementation.

You can reach him out on twitter @arslan0644

The image is a promotional graphic for a webinar. At the top, it features a banner with the text "33RD ANNUAL SOFTWARE QUALITY CONFERENCE OCTOBER 12-14, 2015" on the left and "Brewing Quality Software" on the right, accompanied by illustrations of beer mugs and a coffee cup. Below this, the main title "PNSQC Trends in Software Quality Webinar" is displayed in a large, bold font. A play button icon is centered over the title. Underneath the title, it says "Featuring the latest in software quality by our expert panelists including: Alan Ark, Howard Chorney, Lee Copland, and Anna Royzman".

33RD ANNUAL SOFTWARE QUALITY CONFERENCE OCTOBER 12-14, 2015

Brewing Quality Software

PNSQC

Trends in Software Quality Webinar

Featuring the latest in software quality by our expert panelists including: Alan Ark, Howard Chorney, Lee Copland, and Anna Royzman

Latest on WWW.TVFORTESTERS.COM

Sculpting NextGen Testing Minds

- by Varsha Shastri



Much has been said about why college students (Especially computer science students) have misconceptions about a career in Software testing. They go to an extent of believing that their career has no future just because they got into testing. In fact an entire book can be written about the myths of software testing among college grads.

Why students are averse to take up testing

I too was in such a situation about 4 years back. When I got into testing (accidentally), I was under innumerable misconceptions. Same was the situation of all my friends who got into testing. Here I am going to list out some reasons why I and my friends had such misconceptions.

1. Curriculum

We studied a software testing textbook during my engineering which most of us found boring.

It only talked about processes involved in testing. Such as:

- Where testing falls in SDLC with diagram and explanation – Necessary but this should not be the only important aspect that students study
- Process of bug reporting – This could be learnt after joining as an engineer and need not be in a text book as it does not generate any interest.
- Structure of test case – Just the procedure won't help but interesting test cases would!

Also many other theoretical and boring stuff which did not make much sense before starting a career as Software Engineer. And don't even get me started on all theoretical questions that get asked in exam! It only required students to memorize diagrams and explanations and regurgitate them in exam. College students have young minds and a strong aptitude towards problem solving.

Curriculum should contain more interesting aspects of testing like

- Exploratory testing
- Finding corner cases
- Designing test strategies for different scenarios
- A lab where students design and automate tests – Surprisingly, I’ve never seen a lab for software testing in colleges.
- Some case studies of excellent testing efforts where important bugs were uncovered.

2. Seniors give a wrong impression about Software testing

Students desperately want to get into the following:

Artificial intelligence - Probably because some seniors are studying this in their MS.

Bigdata and cloud - Mostly because seniors say it has a “big” future.

Android development - Because seniors developed a car-pooling app in 5th semester.

Literally any role that involves the keyword "Java" because many seniors brag that they are into it.

Many more - Notions and beliefs through second hand information without any understanding.

And Software Testing? Nah! And you know the further reaction from people (read seniors)!

There sure is a strong prejudice which is really baseless.

A legacy needs to be created where senior students who are into testing inspire the next generation to take up software testing as a career. One way this can be done is by organizing testing workshops in colleges by alumni.

We have a long way to go in this direction!

3. Notion that testing roles involve only mundane tasks

Many say that testing roles that are mundane (by following scripted testing instructions) and some might feel that it’s stifling to a creative individual.

But it’s also true that such boring roles are present in every job profile!

A Java developer's role can also be the most boring thing in the world if he is doing only mundane tasks.

What we need to do is, try and find the role that's right for us.

The problem is not about testing/development it’s about finding the right kind of tester/developer role.

There is no one to stop you from testing/developing in a creative way!

There are many testers who are more technical and better problem solvers than many developers. Get over the feeling that anything that's development is good and anything that's testing is bad.

Some ideas to generate interest in Software testing

1. Interesting books which can be part of curriculum /recommended read.

Reading books is the first best way to generate interest in a topic. Reading right books is more important. Different people like different kinds of books.

Here are some books that I personally found great:

- Lessons Learned in Software Testing – Dr. Cem Kaner, James Bach and Brett Pettichord
- Beautiful Testing: Leading Professionals Reveal How They Improve Software –Tim Riley and Adam Goucher
- Testing Computer Software – Dr. Cem Kaner
- How google tests software - James Whittaker
- Test Driven Development by Example - Kent Beck
- And The Tester's Library by Jerry Weinberg

2. Projects students and fresh grads can undertake

Rarely do college students undertake a project in software testing field. More students should be encouraged in this direction. This helps alleviate some false fears regarding testing.

Following are some suggestions from my end:

i. Monkey testing:

Heard of Infinite monkey theorem? It is said that infinite number of monkeys typing for infinite amount of time can reproduce all works of Shakespeare! Such is the power of randomness.

If that's true then performing random operations for extended period of time should cover most test cases. Right?

This is the principle behind monkey testing. You perform large number of random operations on the application under test. Click, drag, enter values etc. - all in random.

Further you can have smart and dumb test monkeys. You can use your creativity to create test monkeys with a range of smartness levels.

For example, a very dumb test monkey may just click on random positions without knowing anything about the objects/controls that it is clicking on. It may not even log the results.

A relatively smarter monkey can click on random positions, know where it's clicking, and log its actions.

But a very smart monkey can click randomly on known objects, expect the consequence of its actions, verify, log results, file bugs etc.

Try to think of 10 levels of smartness you can give to your test-monkey. You might get interested in making one!

"Germlins" is one example of test-monkey. Its code is available on github. You can take a look at it as well.

ii. Code coverage tool:

In simple terms, code coverage tools help in analyzing how much of the code is being tested by our testing.

This is done by installing test probes within the software systems source code. This can be achieved in many ways One way is by inserting a "visited[i]=1" statement in each branch of the source code and then finding out the number of non-visited areas.

Quilt is one tool that measures coverage. It is also open-source so you can take that as an example.

(However code coverage does not show how well tested the app is. 90% of code coverage does not mean your app is 90% tested. It only means, there is 10% of code which is not tested. Do some more research to find out why this is so!)

iii. UI automation tool:

UI Automation sure has too many tools now that use different techniques.

Simplest way is to choose native APIs which can do basic operations of mouse and keyboard actions and then writing wrappers around them by carefully grouping them together.

You can enhance it by adding record and replay, picture comparison, sophisticated reporting and so on. It can be a very good 4-5 month project.

There are books on how to make your own automation tool.

One example is: Effective GUI Testing Automation: Developing an Automated GUI Testing Tool by Kanglin Li and Mengqi Wu.

If you are confused about how to start off, read such a book and you'll have your roadmap ready.

iv. Profiling tool:

Profiling is basically dynamic program analysis.

It could be space, complexity etc. Usually it is used to help developers optimize the code.

Basic profiling can be achieved by placing a counter in some most used functions/instructions. That way you get the info of where the program is spending most of its time.

There are very complex profilers available today.

However, for a college project, if you can profile memory and speed, then it's more than enough.

v. Code tester tool:

This is basically a code executer with test cases executed along.

It should run the code with as many test cases as possible and find out which test cases it fails.

TopCoder, Codechef are examples of what I am talking about.

Most coding contests I have seen are not tested properly. Usually prizes are given to those who complete most problems in least number of time. Also their code is tested by merely trying a couple of test cases manually. A good coder should create code that considers all corner cases. You can create this code tester tool and test the coding contest entries by creating exhaustive test cases. This might change the contest results many times as well.

In this way, practice writing test cases for all coding related stuff and you can go on to become the testing consultant of your college!

vi. Automatic test case generation:

This has been a topic of a lot of research papers in testing field which has led to lot of scholarly articles which explain different ways of doing this. Lot of artificial intelligence has been applied in this area of study.

One can take up a dissertation on this topic alone. If you want to use this topic for your project, there are innumerable ways to do this for different scenarios.

3. Participate/Organize in Testing Challenges Online and in college fests

Students must participate in testing contests to truly experience the pleasure of catching bugs. Check online and you can find tons of testing contests where you can be rewarded as well.

Better yet, consider organizing a testing contest in your college! Most college fests have coding contests and many other technical events but testing contest is not as common.

A testing event can be very creative and varied. It always need not be testing of a mobile/web app, hacking event etc. I had seen an extremely creative treasure-hunt event in which the act of finding bugs in apps would uncover the clue to next stage.

In another event the contestants had to find the actual functionality of tool/app just by inputting values and seeing the output. They also had to find when the program would fail after understanding the functionality of the app. In fact a very challenging black-box testing contest.

You can also have technical junkyard wars contest where contestants are given some executables which they need to find out the functionality by giving inputs and finding outputs and put the executables together to perform a bigger task.

4. Get inspired!

A wise man once said "Genius is 1% inspiration and 99% perspiration".

But if the inspiration is lacking, perspiration is hard to come by. We should not wait for inspiration to come to us. We should go get it! Where ever it is.

Few ways:

- i. Attend software testing conferences
- ii. Watch videos of related to testing - Of course **TV for Testers** nicely does this job!
- iii. Talking to successful, intelligent software testers around you!

After getting inspired, its commitment that matters.

In God we trust, the rest we test!



Varsha Shastri is currently working for Oracle, automating the tests for Oracle BI applications.

Earlier she used to work for National Instruments where she created automation frameworks to test the LabVIEW programming language.

She is always interested in helping students to learn things better.

Other than these, she is also an ardent fan of Indian classical music and loves to sing during my free time and write some Kannada poetry.

Love to
Write?



In the school of Testing

for your better learning & sharing experience

Homework - Making the Best With What You Got...

- by Keith Klain

"A lot of young people no longer see the trades and skilled manufacturing as a viable career, but I promise you, folks can make a lot more potentially with skilled manufacturing or the trades than they might with an art history degree." --- [President Barack Obama](#)

When it comes to useless college degrees, according to the President, I might possibly have hit the lottery. Art. Of all the things you can study at university, art has to be the subject most often associated with useless, navel gazing, impractical pursuits of higher learning. I mean, what could you possibly do with a degree focused on creativity, communicating abstract ideas, and viewing things in their appropriate context?

And according to PayScale, (Figure A) not only did I get a worthless degree I also should land just somewhere above a barista on my earning potential! Starbucks, here I come!

So sarcasm aside why didn't that happen? Why doesn't my resume read like a list of fast food and coffee shops?

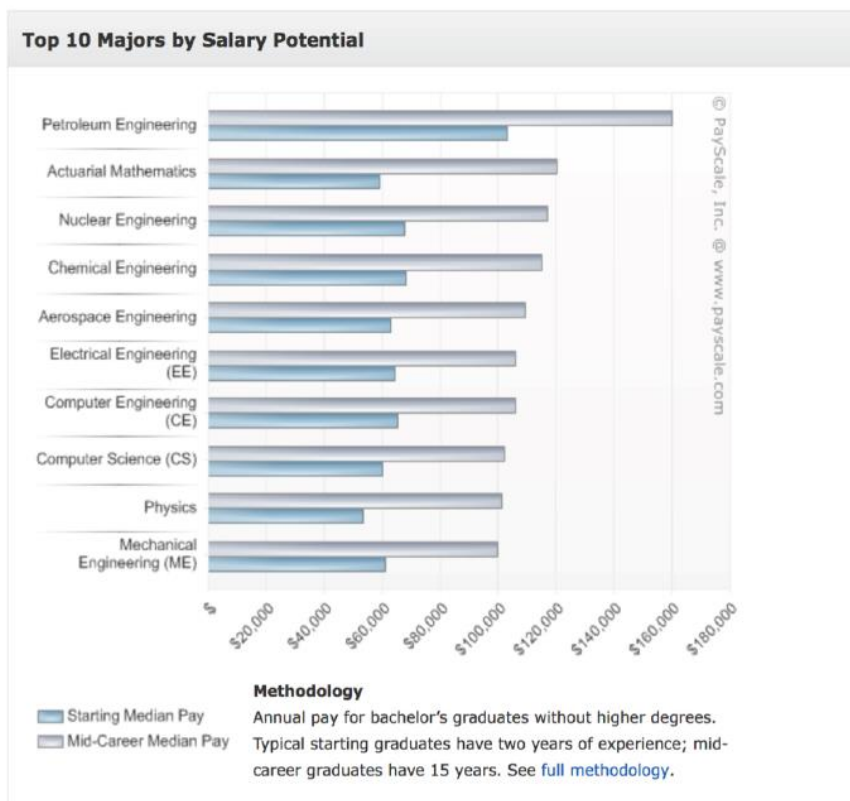


Figure A: Why I Should Be Broke

Looking back on my studies and career, there are common themes to my success (and failure). As well, you can apply those heuristics to other people I've tried to pattern myself after regardless of their degree.

Much has been made recently if getting a [college degree is necessary](#) in the age of information. And although I still believe that having a degree has never "hurt" my career, what you actually DO with that degree is as important as the letters on the diploma.

And as well, as someone who has always felt that educationally, I bring a knife to a gunfight, the most important aspect of these is that they are completely under my control. So to all my fellow art majors and everyone else who feels that their degree (or lack of one) is an obstacle, here is my list of attributes that got me to where I am today.

Attitude

If [Woody Allen](#) was right, and 90% of success is showing up, then I would say the same percentage applies to your attitude. I would rather work with a less skilled or educated person with a great attitude than with a "[smart jerk](#)". I've seen countless PHD level employees or otherwise over---educated candidates that can't figure out why they aren't getting promoted, raises, or just generally more success at work. Almost to a person, the common denominator was a bad attitude.

Knowledge work is hard enough and the problems we are often trying to solve are abstract, so let's not make it worse by being difficult to work with. In my experience a healthy dose of empathy, teamwork, and mucking in and getting on with it go a very long way on the road to success. I've never seen much value in complaining and if perception is reality, creating a reputation as being someone is easy to do and makes a big impact.

Opportunism

I have always considered myself a “dyed--- in---the---wool” opportunist. I volunteered to help start up our college tutoring program even though I wasn’t an education major. **(Figure B)** I said “yes” to my first international move to London having never been there. I agreed to give the overview of our SQM practice performance at the annual company meeting even though I never spoke in public before. I volunteered to pilot building a team in India at UBS.

There are countless other examples of opportunities I either took or created to put myself in a situation to learn or add value to my company. Too often I see people self--select out due to fear when they should take Richard Branson’s advice: “If somebody offers you an amazing opportunity but you are not sure you can do it, say yes – then learn how to do it later!”

Hard Work

So much of our lives is down to nothing more than dumb luck. Where you were born, who your parents were, how much money you had growing up, and a whole bunch of other variables (completely out of your control) contribute a great deal to your chances in life. The longer I’m around and the more people I observe, I come to realize the large part luck has played in my own career. So if so much of it comes down to luck, what part can we play in guiding our own destiny?

That boils down to one thing for me: hard work. How hard I work is one of the few things I have complete control over. I am definitely not the smartest guy you’ll ever meet. I had a State education in a degree that doesn’t do a lot for me. But I will work harder, longer, and do more research than just about anyone you’ll meet. I am determined to do more with the limited resources I have available than the competition, and I’ve seen that this one thing has been the primary differentiator in my career.

Summary

Looking back on my career, I’ve lived abroad for almost 10 years in the UK and Singapore, travelled to 18 countries for business, worked for multiple Fortune 100 firms, and run teams of thousands of testers worth millions in budget. Now I’m the co-CEO of my own company building an outsourcing center in the US and talking about software testing at the White House! Living a charmed life? Maybe, but I like to think I am living proof for all us “art majors” that attitude, opportunism, and hard work are ultimately what’s going to make you stand out from the crowd.

Good luck and keep learning!



Figure B: Evidence of Two Things

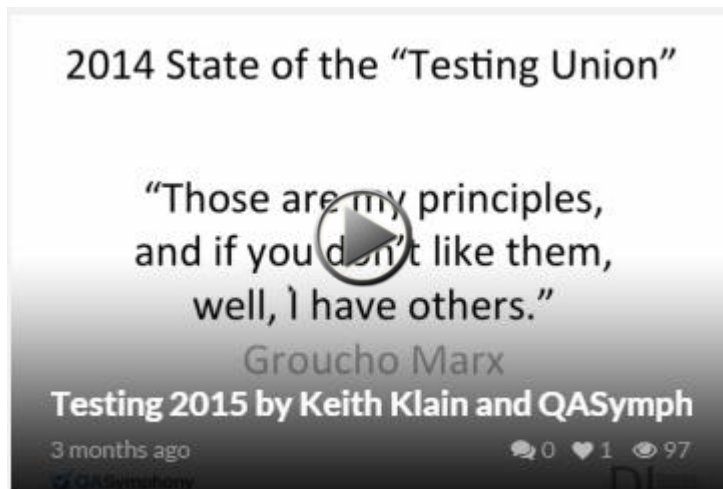


Keith Klain is the co-CEO of Doran Jones, a software engineering and testing consultancy based in New York. With over twenty years of multinational experience in enterprise-wide testing programs, he has built global test teams for financial services and IT consulting firms in the US, UK, and Asia Pacific.

Keith designed the Per Scholas STEP program and is instrumental in its growth and continued expansion. Keith was the recipient of the 2013 Software Test Professionals Luminary award and is the Executive Vice President for the Association for Software Testing.

Reach out to Keith on twitter @KeithKlain or visit his website <http://www.qualityremarks.com>

Don't miss some of the awesome talks by Keith. Go, grab them on <http://tvfortesters.com>



TV for Testers

Your one stop shop for all software testing videos



Exploratory Testing 3.0

- by James Bach
& Michael Bolton

[Authors' note: Others have already made the point we make here: that exploratory testing ought to be called testing. In fact, Michael **said that about tests** in 2009, and James wrote a blog post in 2010 that **seems to say that about testers**. Aaron Hodder **said it quite directly** in 2011, and so did **Paul Gerrard**. While we have long understood and taught that all testing is exploratory (here's **an example** of what James told one student, last year), we have not been ready to make the rhetorical leap away from pushing the term "exploratory testing." Even now, we are not claiming you should NOT use the term, only that it's time to begin assuming that testing means exploratory testing, instead of assuming that it means scripted testing that also has exploration in it to some degree.]

In the beginning, there was testing. No one distinguished between exploratory and scripted testing. Jerry Weinberg's 1961 chapter about testing in his book, *Computer Programming Fundamentals*, depicted testing as *inherently* exploratory and expressed caution about formalizing it. He wrote, "It is, of course, difficult to have the machine check how well the program matches the *intent* of the programmer without giving a great deal of information about that intent. If we had some simple way of presenting that kind of information to the machine for checking, we might just as well have the machine do the coding. Let us not forget that complex logical operations occur through a combination of simple instructions executed by the computer and not by the computer logically deducing or inferring what is desired."

Jerry understood the division between human work and machine work. But, then the formalizers came and confused everyone. The formalizers—starting officially in 1972 with the publication of the first testing book, *Program Test Methods*—focused on the forms of testing, rather than its essences. By forms, we mean words, pictures, and strings of bits, data files, tables, flowcharts and other explicit forms of modeling. These are things that we can see, read, point to, move from place to place, count, store, retrieve, etc. It is tempting to look at these artifacts and say “Lo! There be testing!” But testing is not in any artifact. Testing, at the intersection of human thought processes and activities, *makes use of* artifacts. Artifacts of testing without the humans are like state of the art medical clinics without doctors or nurses: at best nearly useless, at worst, a danger to the innocents who try to make use of them.

We don’t blame the innovators. At that time, they were dealing with shiny new conjectures. The sky was their oyster! But formalization and mechanization soon escaped the lab. Reckless talk about “test factories” and poorly designed IEEE standards followed. Soon all “respectable” talk about testing was script-oriented. Informal testing was equated to unprofessional testing. The role of thinking, feeling, communicating humans became displaced.

James joined the fray in 1987 and tried to make sense of all this. He discovered, just by watching testing in progress, that “ad hoc” testing worked well for finding bugs and highly scripted testing did not. (Note: We don’t mean to make this discovery sound easy. It wasn’t. We do mean to say that the non-obvious truths about testing are in evidence all around us, when we put aside folklore and look carefully at how people work each day.) He began writing and speaking about his experiences. A few years into his work as a test manager, mostly while testing compilers and other developer tools, he discovered that Cem Kaner had coined a term—“exploratory testing”—to represent the opposite of scripted testing. In that original passage, just a few pages long, Cem didn’t define the term and barely described it, but he was the first to talk directly about designing tests while performing them.

Thus emerged what we, here, call **ET 1.0**.

(See [The History of Definitions of ET](#) for a chronological guide to our terminology.)

ET 1.0: Rebellion

Testing with and without a script are different experiences. At first, we were mostly drawn to the quality of ideas that emerged from unscripted testing. When we did ET, we found more bugs and better bugs. It just felt like better testing. We hadn’t yet discovered why this was so. Thus, the first iteration of exploratory testing (ET) as rhetoric and theory focused on escaping the straitjacket of the script and making space for that “better testing”. We were facing the attitude that “Ad hoc testing is uncontrolled and unmanageable; something you *shouldn’t* do.” We were pushing against that idea, and in that context ET was a special activity. So, the crusaders for ET treated it as a technique and advocated using that technique. “Put aside your scripts and look at the product! Interact with it! Find bugs!”

Most of the world still thinks of ET in this way: as a technique and a distinct activity. But we were wrong about characterizing it that way. Doing so, we now realize, marginalizes and misrepresents it. It was okay as a start, but thinking that way leads to a dead end. Many people today, even people who have written books about ET, seem to be happy with that view.

This era of ET 1.0 began to fade in 1995. At that time, there were just a handful of people in the industry actively trying to develop exploratory testing into a discipline, despite the fact that all testers unconsciously or informally pursued it, and always have. For these few people, it was not enough to leave ET in the darkness.

ET 1.5: Explication

Through the late '90s, a small community of testers beginning in North America (who eventually grew into the worldwide Context-Driven community, with some jumping over into the Agile testing community) was also struggling with understanding the skills and thought processes that constitute testing work in general. To do that, they pursued two major threads of investigation. One was Jerry Weinberg's humanist approach to software engineering, combining systems thinking with family psychology. The other was Cem Kaner's advocacy of cognitive science and Popperian critical rationalism. This work would soon cause us to refactor our notions of scripted and exploratory testing. Why? Because our understanding of the deep structures of testing itself was evolving fast.

When James joined ST Labs in 1995, he was for the first time fully engaged in developing a vision and methodology for software testing. This was when he and Cem began their fifteen-year collaboration. This was when Rapid Software Testing methodology first formed. One of the first big innovations on that path was the introduction of guideword heuristics as one practical way of joining real-time tester thinking with a comprehensive underlying model of the testing process. Lists of test techniques or documentation templates had been around for a long time, but as we developed vocabulary and cognitive models for skilled software testing in general, we started to see exploratory testing in a new light. We began to compare and contrast the important structures of scripted and exploratory testing and the relationships between them, instead of seeing them as activities that merely felt different.

In 1996, James created the first testing class called "Exploratory Testing." He had been exposed to design patterns thinking and had tried to incorporate that into the class. He identified testing competencies.

Note: During this period, James distinguished between exploratory and ad hoc testing—a distinction we no longer make. ET is an ad hoc process, in the dictionary sense: ad hoc means "to this; to the purpose". He was really trying to distinguish between skilled and unskilled testing, and today we know better ways to do that. We now recognize unskilled ad hoc testing as ET, just as unskilled cooking is cooking, and unskilled dancing is dancing. The value of the label "exploratory testing" is simply that it is more descriptive of an activity that is, among other things, ad hoc.

In 1999, James was commissioned to define a [formalized process of ET for Microsoft](#). The idea of a "formal ad hoc process" seemed paradoxical, however, and this set up a conflict which would be resolved via a series of constructive debates between James and Cem. Those debates would lead to what we here will call *ET 2.0*.

There was also progress on making ET more friendly to project management. In 2000, inspired by the work for Microsoft, James and Jon Bach developed "[Session-Based Test Management](#)" for a group at Hewlett-Packard. In a sense this was a generalized form of the Microsoft process, with the goal of creating a higher level of accountability around informal exploratory work. SBTM was intended to help defend exploratory work from compulsive formalizers who were used to modeling testing in terms of test cases. In one sense, SBTM was quite successful in helping people to recognize that exploratory work was entirely manageable. SBTM helped to transform attitudes from "don't do that" to "okay, blocks of ET time are *things* just like test cases are *things*."

By 2000, most of the testing world seemed to have heard *something* about exploratory testing. We were beginning to make the world safe for better testing.

ET 2.0: Integration

The era of ET 2.0 has been a long one, based on a key insight: the [*exploratory-scripted continuum*](#). This is a sliding bar on which testing ranges from completely exploratory to completely scripted. All testing work falls somewhere on this scale. Having recognized this, we stopped speaking of exploratory testing as a technique, but rather as an approach that applies to techniques (or as Cem likes to say, a “style” of testing).

We could think of testing that way because, unlike ten years earlier, we now had a rich idea of the skills and elements of testing. It was no longer some “creative and mystical” act that some people are born knowing how to do “intuitively”. We saw testing as involving specific structures, models, and cognitive processes other than exploring, so we felt we could separate exploring from testing in a useful way. Much of what we had called exploratory testing in the early 90’s we now began to call “freestyle exploratory testing.”

By 2006, we settled into a simple definition of ET, *simultaneous learning, test design, and test execution*. To help push the field forward, James and Cem convened a meeting called the Exploratory Testing Research Summit in January 2006. (The participants were James Bach, Jonathan Bach, Scott Barber, Michael Bolton, Elisabeth Hendrickson, Cem Kaner, Mike Kelly, Jonathan Kohl, James Lyndsay, and Rob Sabourin.) As we prepared for that, we made a disturbing discovery: every single participant in the summit agreed with the definition of ET, but few of us agreed on what the definition actually *meant*. This is a phenomenon we had no name for at the time, but is now called *shallow agreement* in the CDT community. To combat shallow agreement and promote better understanding of ET, some of us decided to adopt a more evocative and descriptive definition of it, proposed originally by Cem and later edited by several others: “a style of testing that emphasizes the freedom and responsibility of the individual tester to continually optimize the quality of his work by treating test design, test execution, test result interpretation, and learning as mutually supporting activities that continue in parallel throughout the course of the project.” Independently of each other, Jon Bach and Michael had suggested the “freedom and responsibility” part to that definition.

And so we had come to a specific and nuanced idea of exploration and its role in testing. Exploration can mean many things: searching a space, being creative, working without a map, doing things no one has done before, confronting complexity, acting spontaneously, etc. With the advent of the continuum concept (which James’ brother Jon actually called the “tester freedom scale”) and the discussions at the ExTRS peer conference, we realized most of those different notions of exploration are already central to testing, in general. What the adjective “exploratory” added, and how it contrasted with “scripted,” was the dimension of agency. In other words: *self-directedness*.

The full implications of the new definition became clear in the years that followed, and James and Michael taught and consulted in Rapid Software Testing methodology. We now recognize that by “exploratory testing”, we had been trying to refer to *rich, competent testing that is self-directed*. In other words, in all respects other than agency, skilled exploratory testing is not distinguishable from skilled scripted testing. Only agency matters, not documentation, nor deliberation, nor elapsed time, nor tools, nor conscious intent. You can be doing scripted testing without any scrap of paper nearby (scripted testing does not require that you follow a literal script). You can be doing scripted testing that has not been in any way pre-planned (someone else may be telling you what to do in real-time as they think of ideas). You can be doing scripted testing at a moment’s notice (someone might have just handed you a script,

or you might have just developed one yourself). You can be doing scripted testing with or without tools (tools make testing different, but not necessarily more scripted). You can be doing scripted testing even unconsciously (perhaps you feel you are making free choices, but your models and habits have made an invisible prison for you). *The essence of scripted testing is that the tester is not in control, but rather is being controlled by some other agent or process.* This one simple, vital idea took us years to apprehend!

In those years we worked further on our notions of the special skills of exploratory testing. James and Jon Bach created the Exploratory Skills and Tactics reference sheet to bring specificity and detail to answer the question “what *specifically* is exploratory about exploratory testing?”

In 2007, another big slow leap was about to happen. It started small: inspired in part by a book called [*The Shape of Actions*](#), James began distinguishing between processes that required human judgment and wisdom and those which did not. He called them “sapient” vs. “non-sapient.” This represented a new frontier for us: systematic study and development of tacit knowledge.

In 2009, Michael followed that up by distinguishing between testing and checking. Testing cannot be automated, but checking can be completely automated. Checking is embedded within testing. At first, James objected that, since there was already a concept of sapient testing, the distinction was unnecessary. To him, checking was simply non-sapient testing. But after a few years of applying these ideas in our consulting and training, we came to realize (as neither of us did at first) that checking and testing was a better way to think and speak than sapience and non-sapience. This is because “non-sapience” *sounds like* “stupid” and therefore it sounded like we were condemning checking by calling it non-sapient.

Do you notice how fine distinctions of language and thought can take years to work out? These ideas are the tools we need to sort out our practical decisions. Yet much like new drugs on the market, it can sometimes take a lot of experience to understand not only benefits, but also potentially harmful side effects of our ideas and terms. That may explain why those of us who’ve been working in the craft a long time are not always patient with colleagues or clients who shrug and tell us that “it’s just semantics.” It is our experience that semantics like these mean the difference between clear communication that motivates action and discipline, and fragile folklore that gets displaced by the next swarm of buzzwords to capture the fancy of management.

ET 3.0: Normalization

In 2011, sociologist Harry Collins began to change everything for us. It started when Michael read [*Tacit and Explicit Knowledge*](#). We were quickly hooked on Harry’s clear writing and brilliant insight. He had spent many years studying scientists in action, and his ideas about the way science works fit perfectly with what we see in the testing field.

By studying the work of Harry and his colleagues, we learned how to talk about the difference between tacit and explicit knowledge, which allows us to recognize what can and cannot be encoded in a script or other artifacts. He distinguished between [*behaviour \(the observable, describable aspects of an activity\) and actions \(behaviours with intention\)*](#) (which had inspired James’ distinction between sapient and non-sapient testing). He untangled the differences between [*mimetic actions \(actions that we want to copy and to perform in the same way every time\)*](#) and [*polimorphic actions \(actions that we must vary in order to deal with social conditions\)*](#); in doing that, he helped to identify the extents and limits of automation’s power. He wrote a book (with Trevor Pinch) about [*how scientific knowledge is constructed*](#); another (with Rob Evans) about [*expertise*](#); yet another about [*how scientists decide to evaluate a specific experimental result*](#).

Harry's work helped lend structure to other ideas that we had gathered along the way.

- [McLuhan's ideas about media and tools](#)
- Karl Weick's work on [sensemaking](#)
- Venkatesh Rao's notions of tempo which in turn pointed us towards James C. Scott's notion of [legibility](#)
- The realization (brought to our attention by an innocent question from a tester at Barclays Bank) that the "exploratory-scripted continuum" is actually [the "formality continuum."](#) In other words, to formalize an activity means to make it more scripted.
- The realization of the important difference between spontaneous and deliberative testing, which is the degree of reflection that the tester is exercising. (This is not the same as exploratory vs. scripted, which is about the degree of agency.)
- [The concept of "responsible tester"](#) (defined as a tester who takes full, personal, responsibility for the quality of his work).
- The advent of [the vital distinction between checking and testing](#), which replaced need to talk about "sapience" in our rhetoric of testing.
- The subsequent redefinition of the term "testing" within the [Rapid Software Testing namespace](#) to make these things more explicit (see below).

About That Last Bullet Point

ET 3.0 as a term is a bit paradoxical because what we are working toward, within the Rapid Software Testing methodology, is nothing less than the deprecation of the term "exploratory testing."

Yes, we are retiring that term, after 22 years. Why?

Because *we now define all testing as exploratory*. Our definition of testing is now this:

"Testing is the process of evaluating a product by learning about it through exploration and experimentation, which includes: questioning, study, modeling, observation and inference, output checking, etc."

Where does scripted testing fit, then? By "script" we are speaking of any control system or factor that influences your testing and lies outside of your realm of choice (even temporarily). This does not refer only to specific instructions you are given and that you must follow. Your biases script you. Your ignorance scripts you. Your organization's culture scripts you. The choices you make and never revisit script you.

By defining testing to be exploratory, scripting becomes a guest in the house of our craft; a potentially useful but foreign element to testing, one that is interesting to talk about and apply as a tactic in specific situations. An excellent tester should not be complacent or dismissive about scripting, any more than a

lumberjack can be complacent or dismissive about heavy equipment. This stuff can help you or ruin you, but no serious professional can ignore it.

Are you doing testing? *Then you are already doing exploratory testing.* Are you doing scripted testing? If you're doing it responsibly, you are doing *exploratory testing with scripting* (and perhaps with checking). If you're *only* doing "scripted testing," then you are just doing *unmotivated checking*, and we would say that you are not really testing. You are trying to behave like a machine, not a responsible tester.

ET 3.0, in a sentence, is the demotion of scripting to a technique, and the promotion of exploratory testing to, simply, *testing*.



James Marcus Bach is a software tester, author, trainer and consultant. He is a proponent of testing as an exploratory process and a founder of the Context-Driven school of software testing. He developed, with his brother Jon, Session-Based Test Management.

Lessons Learned in Software Testing, a book he co-authored, has been cited over 130 times according to Google Scholar, and several of his articles have been cited dozens of times including his work on heuristics for testing and in opposition to the Capability Maturity Model. He was briefly a columnist for IEEE Computer magazine.

Today, he works with project teams and individual engineers to help them learn to do the testing that allows them to understand and control the risks of product failure.

Visit <http://www.satisfice.com/> to know more about James or reach out to him on Twitter @jamesmarcusbach

Michael Bolton is a consulting software tester and testing teacher who helps people to solve testing problems that they didn't realize they could solve. He is the co-author (with senior author James Bach) of Rapid Software Testing, a methodology and mindset for testing software expertly and credibly in uncertain conditions and under extreme time pressure. Michael has 25 years of experience testing, developing, managing, and writing about software. For the last 15 years, he has led DevelopSense, a Toronto-based testing and development consultancy. Prior to that, he was with Quarterdeck Corporation for eight years, during which he managed the company's flagship products and directed project and testing teams both in-house and around the world.

Contact Michael at michael@developsense.com, on Twitter @michaelbolton, or through his Web site, <http://www.developsense.com>.

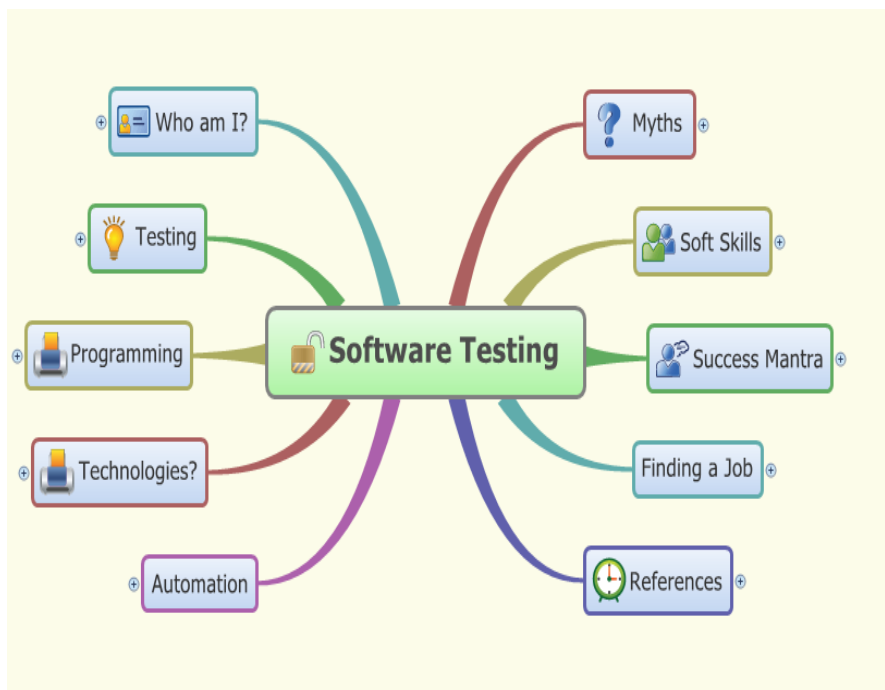


What does it mean to be a Tester ?

- by Parimala Hariprasad

Immediately into graduation, students think that programming is THE VALUABLE JOB and nothing else. Few students who are in contact with the IT industry get tidbits about how things work – through the eyes of the people who tell them how things work. Little do these students know that in most cases, people are pushing down their own perceptions, down student's throat, which students are swallowing happily.

Being a tester means different things to different people. For a college grad, being a tester is having a low profile job that goes only to programmer rejects and poorly skilled people. For beginners, tester means any other guy who is least respected in software development lifecycle. For test managers, tester's means leading a team of de-motivated people testing products/services which they clearly dislike. For some others, testers are skilled people, they are multi-talented individuals and they are highly knowledgeable people. One needs to understand testing to be able to say what it means to be a tester.



INTRODUCTION TO SOFTWARE TESTING

In this article, I will attempt to educate a little on testing and a few general skills. Practicing these might give a fair idea of what testing is and whether it will fit into your career aspirations or not.

Software Testing Myths

"Testing is questioning a product in order to evaluate it" as quoted by Master James Bach. Testing is a mindset and a skillset. Testing is one area where a gamut of skills like technology, soft skills and others are a must. However, software testing field is engulfed with several myths. Let's look at some of them below:

9 to 5 job

Testing is not a 9 to 5 job. Testing requires lot of effort. It requires constant collaboration with rest of the teams in SDLC to facilitate good delivery. Mission critical and life critical projects require round the clock testing support which means many teams stretch beyond regular working hours, and also work on weekends. Some test teams even work in shifts. Testing is a skilled job and testers may have to work till 10 PM all Friday nights to discover the depth of this skill. Testing is hard; it's not an easy job.

Meant for lesser intelligent souls

In some countries, graduates who score lesser marks in technical and general aptitude tests are allocated for testing projects. While this happens, it doesn't mean that everyone who scored fewer marks is less intelligent in any way or that grads who scored more are the smartest technologists. It's unfair to blame testers as programmer rejects. Someone choosing coding may not make them tester-rejects either. Testing needs skills like lateral thinking, investigation skills, critical skills and so forth. These skills don't come easy, they grow with deliberate practice.

Low Pay

Its true testers are underpaid in several organizations, but it's also true that many organizations pay testers more than their programmers for the domain knowledge and subject matter expertise they possess. Everyone will be paid based on their skills and demand for those skills.

No reputation

Testers don't get adequate respect in some organizations. Notice the word **some**, not all. This myth is common because of some troublemakers sensationalizing a few incidents where testers are not valued. Additionally, such cases hide the fact that testers might have not been credible in the first place in such incidents. Such things lead graduates to think that testers have no reputation, hence a poor career option. As long as testers are skilled and do credible work, there is no dearth for reputation.

How testing mindset can be developed prior to getting into testing?

If you are still with me, then it means you are curious. Some graduates willing to experiment with testing can do the following:

Learn about Testing

Understanding the product

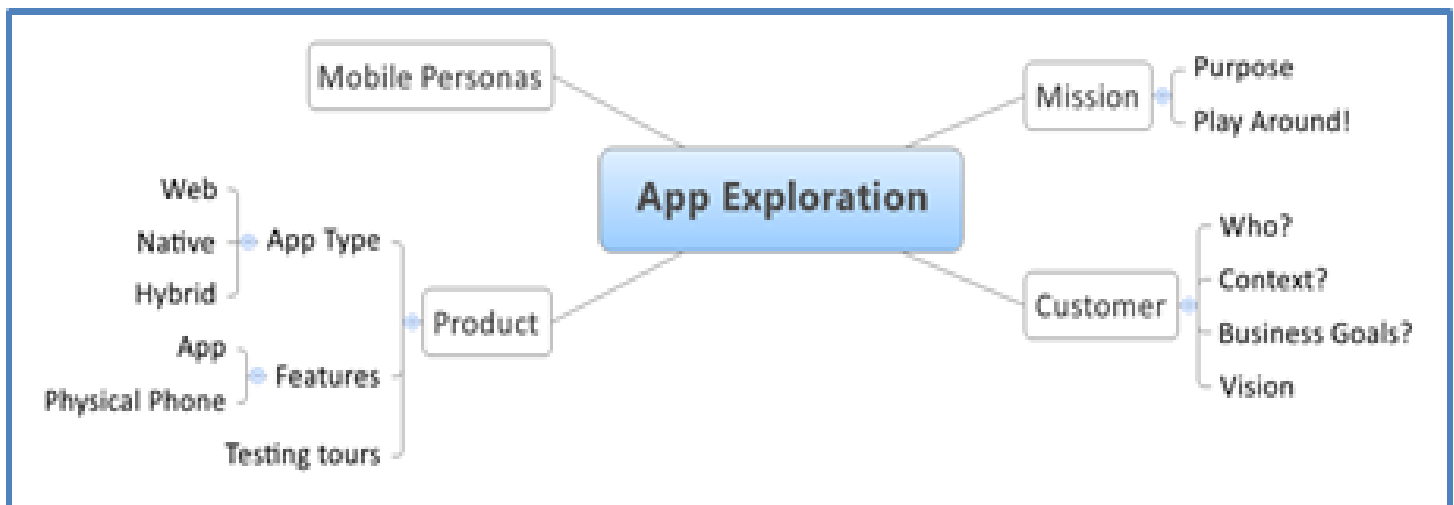
One good way to learn testing is by observing how you use many products in your day to day life. For e.g. how you use mobile apps, what kind of mobile apps you like, what kind of problems you face, how do technical support guys address your complains and so forth.

Product or application under test is one of the important factors to learn about on a new project. What problems or unmet needs is the product solving, how is it solving the problem, is the product sticky enough for users to try out, what kind of users use this product – consumers or enterprises, what is the revenue model for the product.

Exploring the product/app

Exploring individual apps is a critical beginning to testing applications. Playing with the app familiarizes the user to different features in the app. This app exploration exercise helps to:

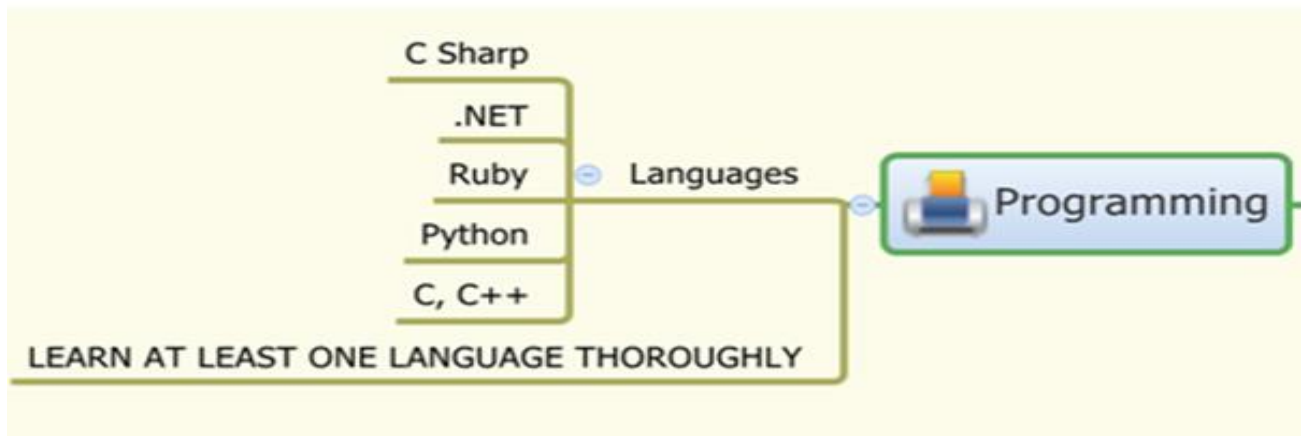
- Know about the app itself and the purpose thereof
- Know about similar apps
- Understand what makes an app popular
- Identify how physical features on the phone are integrated with the app
- Learn by having fun



Technical Skills

Technical expertise is one skill that sets apart any extraordinary tester from an ordinary one. It is imperative to learn at least one programming language from scratch. Knowing how code works internally, what compilers do and how application/system programs function help in designing better technical tests for testing the product. Additionally, learning underlying technologies used to build the product, domain knowledge; target user segment can help create better scenarios to test the application. It's important

for graduates to learn at least one programming language whose concepts can be applied to testing any product.



Every project demands select set of technical and soft skills. Many testers take it for granted that testing is a common skill that is applicable on any project or domain. This assumption drives them to think that they have already arrived with all skills. It is important to draw a simple map of what skills may be needed – like new domains, technologies, tools and spend at least 20 minutes a day on learning them. There are scores of free courses on Udemy, Coursera and Khan Academy that can help. Even stalwarts like Satya Nadella spend some time every month learning a new concept. Learning is a lifelong journey and a critical one at that.

Soft Skills

Some projects may need testers to interact with customers on a regular basis which in turn demands them to be aware of several soft skills like leadership, verbal communication, client communication, email etiquette, presentation skills amongst others. Soft skills are harder to learn and implement, but nevertheless, critical for product success.

Seeking Help

Knowing that you need help and seeking the help you need is a very important aspect of learning. If you intend to solve all problems by yourselves or spend several days before approaching someone for help, then you are setting up yourselves for failure. If you need help, you must ASK FOR HELP.

Here is a simple heuristic to ask for help. When you hit upon a problem, understand the problem deeply, come up with a few solutions and apply those solutions. Give it your best. If it still doesn't work, reach out for help. When you ask for help, state the problem, emphasize on the steps you took to solve the problem and tell them how you failed. This will demonstrate to the problem solver that you care about

the problem. From thereon, any person will be interested to solve your problem unless it's your bad day or the person on the other side is a maniac ☺

Emotional Labor

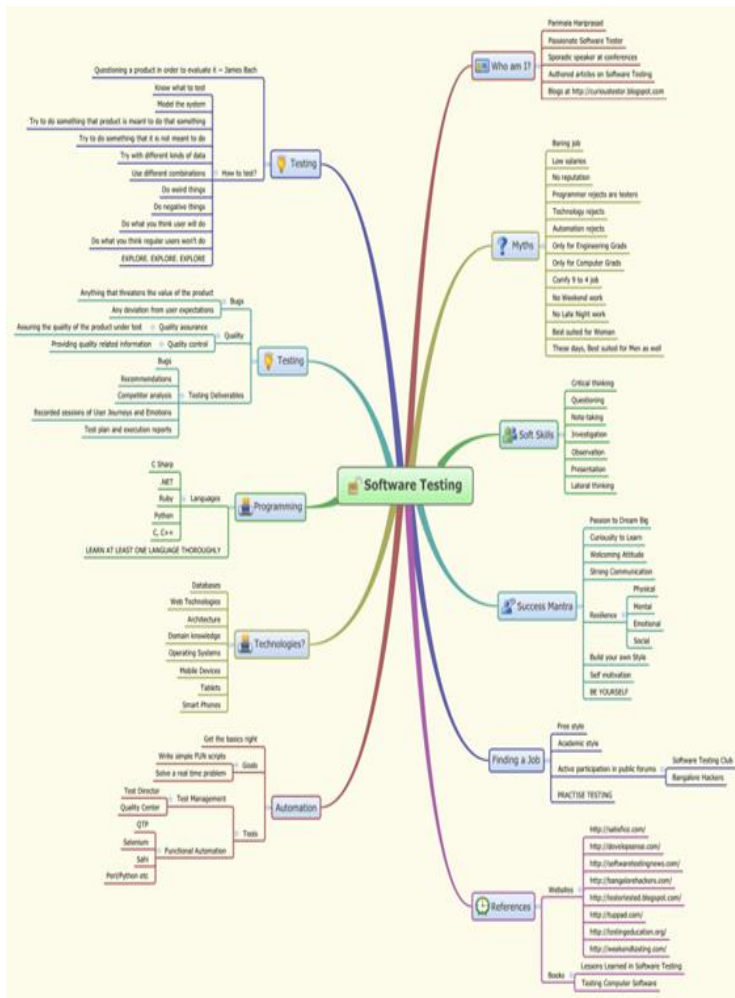
According to Seth Godin, "Emotional Labor is doing the difficult work of bringing your very best self to each interaction braving pain." It is to do the work even when you don't feel like it. Expending emotional labor is a quality that is expected of everyone because they no longer do what they love most. People with a gift for emotional labor are called Linchpins.

Linchpins hope for a better world. They already live in the world that they are about to create. They slog it out, not knowing that they'll become indispensable. Linchpins make changes happen. Every tester must ask them, "Am I a Linchpin? Am I capable of expending emotional labor?"



Mindmap below summarizes the list of activities one can do to get started in testing. What does it take for anyone to succeed in testing? A combination of all factors mentioned in the mindmap below plus a Drive to accomplish personal goals. It takes a lot of hard work, patience and perseverance to build credibility as testers.

Do you have any questions? Email them to Parimala Hariprasad at Parimala.shankaraiah@gmail.com



Parimala Hariprasad spent her youth studying people and philosophy, and in the workplace she applied her learnings to helping create skilled testers. Parimala has been a tester for more than eleven years in domains of CRM, security, e-commerce, and healthcare. Her expertise lies in test coaching, test delivery excellence, and creating great teams that ultimately fire her because they become self-sufficient. Parimala has experienced the transition from web to mobile and emphasizes the need for design thinking in testing. She frequently rants on her blog Curious Tester, tweets as @CuriousTester, and is on LinkedIn.

[Click on the image to view on the web]

SmartBear Webinar:

APIs - Testing the magic behind the scenes

Date: March 14, 2012

Run Time - 1:01:40

© 2012 SmartBear Software

Teaching new testers about what it means to be a Tester

- by Martin Nilsson

Software testing is a very complex craft that requires a high level of skills in a broad set of areas, even more than that required in most professions. Not only do the tester needs to understand the technology she is working with, but she also needs to understand the context with regards to the customer, the business needs, the co-workers, the processes, the methodologies to software development as well as for testing. And on top of that, the tester must be able to communicate the information found to different stakeholders.

My name is Martin Nilsson and I have worked with testing in several different fields for many years. Currently I, together with my colleagues Erik Brickarp and Maria Kedemo, am developing and teaching a professional test education in Sweden. It is a type called Yrkeshögskola and this particular one spans over one and a half year and is based on the Context Driven Test-principles. In this article I will share what it means to be a tester and how my students are taught testing.

The Definitions

The first few things I taught my students about software testing were the following definitions of testing:

"Questioning a product in order to evaluate it" by James Bach and

"Testing is an Empirical technical investigation done to provide stakeholders, information about quality of a product or a service" by Dr. Cem Kaner.

I like them especially because it forces the tester to actually think and investigate. If no questioning of the product has been done then no testing has been done. Using these definitions also means that a computer can never do testing because a computer program can never question a product or do an investigation, it can only follow directives (this can of course also bring benefit but of another kind as compared to testing).

Testing as a Contextual Practice

The second thing I taught them about software testing was that it's a highly contextual practice because of the extremely broad area that it covers. Today, software is found everywhere, in clocks, lightbulbs, cars, regular pc-programs and the software is developed with different purposes, for different users by different people.

The good thing about it is that whatever you desire to work with, there is a place for you in testing. There is a place for you if you like coding, if you like solving problems, if you like working on the social side of software development, if you like to document things, if you like to bring order out of chaos, if you like navigating in chaos, if you like math, if you are great at languages or if you like to understand how user interfaces can help people to use a product. On top of that, everything around a tester is constantly changing; the technology is evolving, new tools are created, new ways to organize development come into fashion and people from different countries and cultures become your colleagues.

I would like to share one experience while I was working at Ericsson in Sweden. We had a well-oiled machinery that was creating great platforms for mobile phones. But a year after I was hired, we transitioned from a waterfallish to a more agile way of working using Scrum. Shortly thereafter the first iPhone was released and we basically had to throw out all our work in progress, rethink and start working with new technology in a reshaped world of telecom. Soon after that we got merged with company called ST and suddenly our developers were not found in Lund, Sweden, anymore but in Greater Noida, outside of Delhi. Within a relatively short time my ability to learn and adapt and understand new technology and new ways of working was challenged to my limits. During this time I also learned the rules of cricket and started to read Times of India on the web to help form better relationships with my new colleagues. And I can proudly say that I made several great friends over there. And I have been invited to several weddings! My colleagues who were used to work with the same tools year after year, needed exact instructions on what to do and had no desire to educate themselves or evolve their skills; struggled during that fast occurring transitions and were unable to provide value to the projects. Point is, your ability and desire to learn continuously, matters.

Testing as an Information Service

My friend Dawn Heynes taught an important lesson to my students that "Testing is an information service".

We provide information about the product to different stakeholders. An obvious stakeholder is a developer who needs to understand how the code works and affects the product. But there are also other stakeholders such as product leader (who needs to understand how the development is moving forward

and the current issues), the product owner (is the product solving the problems for the intended customer?), the customer support team (what should they know if a customer complains), sales people (what can and cannot be demonstrated) and possibly many more. They all need different form of information and that is something that a good tester can provide. And it's not only bugs that are interesting pieces of information since they can in some cases be irrelevant.

Take Minecraft for example; it's the game that got the small game development company Mojang sold to Microsoft for 2 billion USD. People had paid for the beta version way before the game was finished. Obviously they were willing to pay despite the game being in early stages filled with bugs and missing features. For those people there was different quality criteria that was more important than a bug free software.

Skills that matter

The contextual nature of testing and the complex environment a tester works was a challenge when designing the test education course. It would be very difficult to cover every topic even when having one and a half year to do it. So we had to focus on the essentials of testing; what does it take to be a good tester and what should a good tester learn. My boss, Henrik Andersson, developed the general direction of the course and he figured out that when everything that is changing around a tester is taken out of the equation then this remains:

- A tester has to be curious
- Should be able to learn (and preferably quickly)
- Tester must be able to analyze a problem from multiple viewpoint to understand what's really the problem
- A tester must be able to think laterally and most importantly should be able to communicate with different people involved in a project.

In initial months my students studied and learned about the essence of testing and training in lateral and critical thinking. I strongly recommend any aspiring tester to read the books by Jerry Weinberg that my students also read:

- Perfect Software – and other illusions about testing
- Are your lights on?
- A general introduction to systems thinking

Only the first book is specifically about testing but they all speak about the core understanding that a tester should possess i.e. about problem solving as well as about the human role in software development. Every piece of software we create is meant to be used by humans and is developed by humans and therefore a good understanding about humans can greatly improve the work of a tester.

The Result

So how did my students perform after a couple of months into the education? When I handed over a software to my students to test; after the first day of testing all of them provided this type of report:

"After touring the product (learning about the products and it's context), doing an analysis of risks and what quality attributes that I/we believe are most important: this is what I/we have chosen to test, these are the results and this is what might threaten the quality perceived by customer".

Despite lacking deeper knowledge in specific tools or test techniques (which were covered in later courses) they could all do independent work and provide value in the form of information from day one.

And this is important because this is often how it looks like when working as a tester: You get a new assignment with a new technology that requires new tools but if you are able to provide information from day one about the product then that information could be valuable for the right stakeholder. Once you learn more about the product and learn to use tools, your information gets increasingly better and thus you can provide increasingly valuable information.

The Future

What is awaiting my students once they finish their test education? Well, one thing is for sure and that is my students will all work in very different kinds of contexts with different kinds of challenges. They will have to become experts in learning, because every new project in which they find themselves working will impose new kinds of demands on them. They will have to become experts in understanding how we humans work with software and learn how to communicate in different ways with different people.

All combined they can look forward to a really, really enjoyable future. They will meet many interesting people throughout their careers and they will get to work with cutting edge software in a number of different fields. Personally I am having a blast in this field of work and I am confident that my students will too!



Martin Nilsson has had a passion for test even since his eyes opened and he realized that testing is way more complex and interesting than just writing and automating test cases. He actually ended up in testing by mistake several years back when he believed he had applied for another assignment. Five minutes before the interview he double checked the invitation and realized that he was there for a testing position. He had no idea what a tester was supposed to do but somehow he got the job. He likes to refer to that incident as the best mistake he has done in his career and every assignment since then has been more fun and interesting than the previous one.

Martin has been working close to hardware on mobile platforms, he has done performance testing in the shipping industry, he has been conducting system testing with Business Intelligence and he became the test coordinator for an R&D organization with a hundred testers. Lately he has been teaching and developing a one and a half year long test education. Make sure to check out #YHTest on Twitter to get a sneak peak of what is going on there! Testing and the related topics take up a lot of his time and interests but besides that he has a passion for photography and he loves tinkering with microprocessors. But most time by far is currently spent on his two children who are currently testing the world☺.

Master the Essentials of UI Test Automation: Chapter Two

- 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 Two: Before You Start

"Plans are useless, but planning is indispensable." This quote from American President Dwight Eisenhower is one of my favorite quotes. Sure, he's talking about the run up to D-Day in World War II, but it's applicable to so many things in life—especially software development.

You need to ensure you're spending your time and effort wisely before you jump in and start hacking away at automated tests. Here are a few things I've found helpful to work through as part of the planning process.

Why Automate?

Why are you considering bringing in test automation to your delivery process? Take some time to get very specific about the problem you're trying to solve. More importantly, make sure you're tackling a concrete business problem. UI automation's a nifty tool, but it's an expensive one to get adept with, and it's a very costly tool to deal with if you use it badly. It's also only one form of test automation, and by itself it's not going to solve much of any delivery "challenges" you're having.

Here are a few business-related areas in which UI automation may be helpful:

- Long release cycles due to time required for manual regression testing
- Testing falling behind development during release cycles/iterations/sprints
- Rework costs due to regressions of high-value features
- High support costs due to escaped bugs around high-value features
- High cost of testing high-value features against multiple browsers and operating systems
- Exploding cost of testing against multiple mobile platforms and browsers

Worst of all, here are two areas that, for me, trump all other concerns:

- Loss of executive-level trust in the team's ability to deliver good software
- Loss of trust and business from your end users or customers

The last two, especially the last one, should concern every team member. It's really bad if you've lost higher-level management's trust. It's potentially disastrous if you're losing customers.

There are some areas for which you should **not** consider UI automation as a solution:

- Validating cross-browser look and feel
- Guarding against layout and style regressions
- Saving money by cutting number of manual testers

UI test automation may help you solve technical or process problems, but ensure you're first asking *why* and focusing on solving business problems first.

Do You Have the Environment to be Successful?

Once you've decided that UI test automation is a good choice for you, the next critical factor is whether your team is able to be successful at UI automation. Several different aspects come into play:

Do You Have Stakeholder and Sponsor Buy-In?

Adopting UI automation will require significant changes to how you work. You'll need team members with the right skills, you'll need resources (no, "resources" are not people), and you'll need additional time.

Adding UI automation into your delivery process will decrease your velocity. You have to make sure your stakeholders and sponsors truly understand that. They have to support the decrease in velocity for the improvement in delivered business value.

"Decrease in velocity" is always something that concerns the business side of the organization. I've found it very helpful to tie back to the specific business-level issues discussed earlier in this post. Say this:

"Yes, we're going to slow down a bit, but the goal is to cut the support costs we're incurring through escaped bugs. We're also hoping to gain back revenue we've lost from the decline in license and service renewals."

This links your efforts back to the things the stakeholders really care about: the organization's mission and bottom line.

Do You Have the Right Communication?

Good communication is the foundation of every successful human effort, software notwithstanding.

Are your teams able to get good information about features in a timely fashion? Do designers, developers and testers all talk regularly about how UX/UI work is accomplished? Can your testers get early input on UI design to help make testable screens? Do your testers understand what the stakeholders' highest priorities are for each feature, and do they understand the business value behind those features?

If the answers to any of those questions are "No," you will need to address those issues as you move forward with your automation—or suffer the friction and stress that falls out.

Great communication helps ensure everyone involved in UI automation knows the priorities before they start their work. It helps everyone understand how to balance automation work with focused manual testing, and it helps focus automation work on the *right* parts of the system.

Do You Have the Right Team Structure?

UI automation rarely succeeds when testers are expected to create all automation in a silo or walled-off room. I already mentioned the importance of early, frequent communication between developers, stakeholders, testers—basically the entire team.

If your teams are fragmented into highly constrictive silos, you will likely see additional friction and difficulties when trying to clear up basic fundamentals such as getting good locators on elements around which you're building automation scripts.

The best structure for any team is an open environment that empowers frequent communication directly between concerned team members. Co-located teams always function the best; however, geographic dispersion can't be an excuse for poor communication.

Break down communication bottlenecks in your teams. Guide project managers or others to get out of the mindset of requiring communication to flow through them—wheel/spoke communication routed through one person is guaranteed to cause problems as teams try to get more efficient at their work.

Encourage your testers to reach out directly to developers when possible. The developers can clear up issues around tricky asynchronous operations, or other behind-the-scene functionality that's not apparent to someone looking only at the UI.

Setting up a team's structure for success means as few roadblocks and walls to frequent, candid discussions.

What Tools Can You Use?

Notice I've left the actual tooling for last. Yes, yes: the toolset you use is critical, but it doesn't matter what tools you select for automation if you haven't answered the *harder* questions first.

You can finally jump into tool selection once you've addressed that you've got a clear case for why you're going to use automation, and what problems you're trying to solve.

There are a huge range of UI automation test tools available. Some are open-source, some are free and some are commercial. There are also many types of tools, from drivers to frameworks to entire suites.^[1]

Selecting the right toolset for your team means answering a few more questions:

- **What communication mechanisms will you use for tests?** Do you need a grammar-based specification? Will recorded tests with coded steps be clear enough? Are 100-percent coded tests clear enough for everyone?
- **Who writes the tests?** Will testers be solely responsible for test creation? Or, will developers have a hand in it, as well?
- **Who maintains tests?** Will the team that writes the tests maintain them, or do you have an outside contractor writing tests and handing them off to an internal team? If you have two (or more) different teams, make sure the teams have the skills, aptitude and time to take on the selected tool.
- **Who uses the tests?** How will you run your tests? Will the suites be triggered manually? Will they be part of a scheduled suite to be run via a [Jenkins CI server](#) or Team Foundation Server build? You'll need people who can handle the care and feeding of those environments, and you'll need the infrastructure required, too.
- **Who uses test results?** Who in your organization needs what level of information about your tests? Keep in mind that your stakeholders often need one set of data, while your team needs another.
- **Do we have the skills?** Lastly, you'll need team members with the right skillset to build, manage and maintain all the pieces necessary for a successful automation effort. Your team doesn't need those skills right now, but they'll need support to develop those skills in a timely fashion.

Telerik put out a ["Buyer's Guide" in 2014](#) that answers these and other questions.

People and Resources Come Next

All of these high-level planning concepts will be central to your team getting automation started as quickly and effectively as possible. You shouldn't spend months answering these questions—analysis paralysis can cause entire organizations to lose a lot of time wringing hands over silly details.

Spend enough time to get a sense of comfort that you've at least talked about the major points above. After that, it's time to move on and discuss the people and resources you'll need for a successful effort.

What do you think of the planning topics we've laid out here? Are there other topics you've covered in your own automation efforts? Let us know in the comments.

Next Up: People

[1] An automation driver is responsible for driving the UI application around. Think of Selenium WebDriver or the Telerik driver. An automation framework sits atop the driver and normally gives teams a grammar-based approach for writing tests (think Cucumber, Fitness, SpecFlow and so on).

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.



Missed our webinar on 25th March? Worry not. Here is the recoding -

Master the UI Test Automation Essentials

Webinar: Master the Essentials of UI Test Automation

Run a Pilot

- Learn to set expectations
- Learn to communicate earlier
- Learn how your system and your toolset works
- Learn to make system testable
- Adapt your process

Telerik



Sharing is caring! Don't be selfish 😊

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



Do you think that QASymphony's products are quite different than conventional tools in market that typically support traditional way of doing testing? What motivated you to launch a product which is different than conventional tools in market and is based on different testing philosophy?

Yes, I think our tool is very different. We actually have a different philosophy on what the future of testing tools should be. But it's taken 4 years to get to that vision. We're about to roll out the product that is the culmination of that vision -- qMap. It is actually a visual map of the quality of a product or application. We can build a sitemap for any application and present information about the quality of the application, such as defects through color coding. This creates a visual representation of where your team needs to spend their time to get the product ready to go to market. What we're building is a product that can help people know how to spend their time in testing. This goes back to context driven development in that it's the product and how people use it that matters -- not the number of tests you've run. The map we're producing takes all that data and produces it in a way that people can understand it easily.

You have experience of working at almost all roles and levels in software development. What is your opinion about the way industry (at large) currently measures testing with typical metrics?

There are some metrics that are good, but they fall short when you are dealing with a fast release cycle. Those metrics don't necessarily help you make the decisions that matter. If you have defects in the areas of software that no one uses, who cares? But if you have defects in areas that are going to affect the user, that matters greatly. This is why a visual map that shows the most important areas of the product is incredibly useful, and why we're excited about the release of qMap. Visual maps that give you different ways to layer information on top of a map of the application provides a much better tool to make decisions that have a direct impact on the user experience.

In your opinion, what are the testing metrics that matter?

Every team must decide which metrics are going to most directly affect the bottom line of the business. The answer is different for every product. And yet, as every company works to determine the unique metrics that matter most, there is this one universal truth -- the perception of the end user always impacts the success of your business. Testers must be constantly channeling the voice of the customer as they work, looking for ways to improve the product for the people who will ultimately be using it.

Your work and contribution to Vietnam's IT field is highly acknowledged. Would you like to share some memories/experience from your work in Vietnam?

I started my first company in 1995 with the intent build a development center in Vietnam to serve the telecom space. I went to Vietnam at the time at the telecom space was just starting so for the next 14 years I was building that company and another company out of India. Out of these experiences, we built a lot of software and learned much about how to write software well and how quality comes into play at the end of the project when the product gets into the users' hands. These experiences are very special to me and helped build lay the groundwork for the work I am doing today.

What are the skills that you actively look for while hiring new testers?

At the end of the day, we look for people with bright and inquisitive minds that want to solve problems. We look for people that want to want to learn and want to challenge -- they assume whatever is handed to them has a problem and it's their job to go in and find it.

Technical background and skills are a nice-to-have, but ultimately we believe that we can take someone who is bright and willing to learn and train them in a short amount of time to be a very effective tester.

According to you, what does 2015 hold for Software Testing?

2015 continues to be a transition year for software development and software testing. Agile development continues to grow and make its way to larger and larger companies. As larger companies roll out agile testing, they will realize that the old tools and the old approach to testing aren't good enough. That started 1 - 2 years ago, but is accelerating through this year, which is why we're excited about QASymphony. We have the right tools, at the right time, in the middle of the right movement.

Your advice to our readers would be...

Don't ever be satisfied. Challenge yourself. Be inquisitive and be open minded. Learn about what other people are doing and see how you can incorporate what other people are doing in your daily work so you can become a more effective tester, test manager, etc.

My final advice would be piece of advice would be to everyone who is in the software space -- the products we build are designed to make people's lives better, faster etc. Always ask yourself, how is what you're building making people's lives better, how is it going to make their work more efficient? And if you don't know the answer to that question, stop what you're doing until you've figured it out.

What is your opinion about Tea-time with Testers? We would be happy to know your feedback.

I think Tea-time with Testers is a great resource for the testing community. It's important for testers to be able to connect with each other and learn from one another and you do a great job in facilitating that! I'm honored to be a part of this issue. Keep up the great work!

Happiness is....

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



Like our FACEBOOK page for more of such happiness

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



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

Like mysteries? Like to explore? Consider testing as a career

[Author's Note: This being a special issue on "College Grads", this article is about presenting testing from a different viewpoint to attract good minds to a career in testing]

Do you like reading mystery stories? Enjoy the apparently disconnected events pieced together by the detective to explain the connection. Enjoying the tension in each thread and the utter joy when a logical explanation stitches these, releasing the tension. The curiosity that keeps you riveted to the book as you flip every page oblivious of the ticking clock and the changing landscape outside. Love the intense reasoning to arrive at hypotheses and the hard work to prove/disprove these?

If you do, then you should look at testing as a career option. As a student, you probably think of testing as a mechanistic job of scrubbing software that is boring and low on the creativity spectrum. Banish this thought. Visualise a set of underground tunnels dark and damp where demons lurk and your job is to hunt them. Would you think of this as a job akin to cleaning the sewers and go about mechanistically walking down each pipe and brandishing a sword? Certainly not! Analysing situation, hypothesising where the demons might lurk, the thrill of stalking up to them with the right tools/techniques and joy of winning over them, is what testing is all about.

Knowing the various types of demons, what they feed on, where they like to live, how they attack and expose themselves is what testing is about. Good testing demands intellectual firepower to reason and

deduce, it demands grit to check-out the reasoning. The latter is the only one that may involve some rote/repetitive work.

So, you have graduated recently and are considering career options. After an intense few years at the college, you are curled up cozily on the couch with a mystery book. And it is engrossing. As the night gives way into dawn, you yawn! After a few winks the alarm goes. Time to get up and go on a hike & trek up the mountain/forest. Your second love - Exploring Mountain trails.

Of charting out the trails from the map. Going along these and exploring new paths. Getting lost, backtracking, and discovering new paths. Analysing, questioning to understand better, to go closer to the goal. To explore the unknowns, expand your horizons and becoming wiser. The attraction of the enormous vastness out there that lures you in, challenges your senses and intellect. Aah, the joy in exploring.

Testing involves serious exploration. Exploring what is it made up of, how the various elements are interconnected, gives you a good picture of the system. To understand what is expected of the system, you need to explore from the view of the various users- who uses what, how much and when. And then to understand the dangers lurking cautious exploration is necessary. And you thought testing was a rote job? Nah. It is a heady dose of exploration laced with mysteries.

What better combination can you ask for in charting your career? To be able to question, analyse, improvise, and exploit technologies to find bugs, to deepen understanding of the system, usage and of the world. Everything that makes you a fine software engineer. Sharpening the skills, across the full spectrum of software development, be it in the area of analysis, design or code.

The discipline of testing is not about jargons and tools. It is about skillful exploration to understand, intelligent use of techniques and tools to uncover issues and delivering value to end users by enabling them to deliver superior outcomes.

It is a means to become a better software engineer. And over the long years, testing has to be a means to discover the inner beauty and in way the Zen

So, as you consider the options for your career, think about Testing! Bet you will love it especially if you like mysteries and exploration. And one day you will explore your inner self and uncover the beautiful mystery in your soul.

All the very best.



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

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

He can be reached at ash@stagsoftware.com



[Back To Index](#)

testing intelligence

- *its all about becoming an intelligent tester*



an exclusive series by **Joel Montvelisky**

Letter to a Starting Tester

[Editor's note: For the benefit of target audience of this special issue, we are reprinting this masterpiece by Joel Montvelisky]

I've been working on the [State of Testing survey](#) and report for the last couple of months, and as part of this project I've talked to a large number of testers and testing teams.

In some of these talks I explained how 17 years ago I started working as an [accidental tester](#), how I tried to escape from testing during the first 3 or 4 years of my career, and how somewhere along the road and without really noticing I found my testing vocation.

After one of these chats I realized that when I was beginning my work as a rookie tester I really had the need for a mentor to help me get started on this journey. There were many times when I would have appreciated professional guidance and advice, especially during some of my moments of doubt.

And so, I decided to write here the email I would have sent to myself back when I started testing to help me cope with some of the main challenges ahead.

Maybe this email is only cheap therapy for myself, but there is also a chance that it may help some of the testers who are only now starting their professional endeavors.

An email to Joel, a starting tester, back in 1998

Dear Joel,

I wanted to send you this mail to help you during some of the difficult times and the challenges that you will encounter as you start your professional career as a tester. What I will tell you may sound lame and trivial at times, but these are the things you will need to hear (and do) to cope with a number of the situations that await you in your coming career.

No one really knows what you need to do better than you do, in the end you will need to figure it out on your own.

Many times you will feel that you don't understand what's expected from you as a tester.

People want you to find the bugs and test the product, but they don't have time to explain what the product really does and how, they will not be open to criticism, they will also hate it when you bring them bad news, and on top of everything else they also expect you to complete all your work within a couple of minutes...

Even though you did not go through any special training, you are suddenly the expert in testing, and sometimes this new responsibility will weigh too much in your shoulders.

As strange as it sounds, no one in your team knows how to do your work better than you do (this will be especially true in the start-up companies where you will work at the beginning of your career). It will be up to you to learn and figure out your job, and to define your tasks in the best way you can with the resources at your disposal.

Don't count on the knowledge of others in your team to rescue you from your responsibility...

There are other testers out there that you can talk too, look for them and share your knowledge and questions.

You are not alone!

Even if you are the only tester in your company, there are still other companies nearby where you will find other testers.

One of the biggest and most important things you will do is to lose your "public embarrassment" and reach out to other testers, to talk with them about your questions, challenges and dilemmas!

You will be amazed about how similar your issues are, and how much you can learn by simply talking to them and coming up with shared ideas on how to solve your professional predicaments.

This will help you learn that asking questions is not a sign of being weak or dumb, but a sign of being professionally confident and smart...

Testing is as much about learning and asking questions, as it is about pressing buttons and reporting bugs.

Hand-in-hand with the last point, you should also learn to ask the people on your team questions about your product, this will help you become a better tester.

Many times you'll see that a developer comes to you to explain a new feature or a change, and after he/she is done explaining (or at least after they think they are done explaining) you have more questions than the ones you started with. When this happens don't be ashamed to ask more questions and to request this person to explain the point from a different angle or using different examples.

A number of (nice) developers may forget that you are not aware of all the technical details, or they will start their explanations based on other assumptions that are not known to you, or they are simply bad communicators and so explain stuff in the worst possible way...

This is just the way it is, and you don't need to be afraid to keep asking until things are clear enough for you to do your work.

It is as much their jobs to make sure you understand how to test as it is to write the code correctly.

Whenever you get a feeling that something is not right, don't keep quiet! Understand what bothers you and communicate this to the team.

At times you will get a feeling that something is not right with your product or your process, and it will be your instinct to think that it is you who made a mistake during your tests.

This will surely be the case many times, but once you have re-checked your assumptions and your procedures, and if you still have that feeling of something not being right make sure to communicate this to others.

Stand your ground when you think you are right.

Sometimes it may be a matter of interpretation, you think that the feature should behave "this way" but the developer thinks that it should behave "that way".

When this happens go to someone else who will help you make the correct choice. Try someone who knows the customer and will be able to provide feedback based on their knowledge of how the users work.

The same goes for the times when you see your project is going to be delayed, but you see people behaving like everything was normal and OK.

If you see that features are slipping, and that the quality of the deliverables is below the status you expected them to be at this time make sure to raise a flag and wave it for the whole team to see it.

After all it is your job to ensure the quality of the process and not only of the product under test!

You are not the gatekeeper of your product by brute force!

Having said all these, it is not your job to stop the bugs (or the versions containing them) from walking out the door.

Your job is to provide visibility into the status of your product and your project, and give everyone on the team the information they need to make their decisions.

After you have given everyone the correct information they will need to make the choice whether to release the product into the field or not. You may be part of this team making the decision, but your voice will never be the only one that counts!

Remember that you may not have all the information related to marketing, sales, competitors, or a range of other factors that are usually involved in the process of deciding whether to release a version to the field or not.

Have fun...

Testing should not be 100% serious all the time!

It is OK to have fun and to joke around with the people in your company, just remember that there are times for joking and there are times for keeping serious.

And one last thing!

When you are working in the Silicon Valley around 1999, look for a company called Google, and ask them if they need a good tester. Even if they pay you only in stock take the job... It will be worth it!



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.

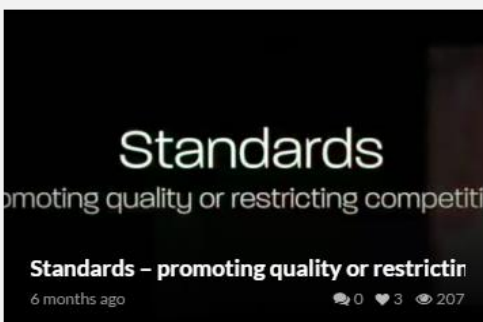
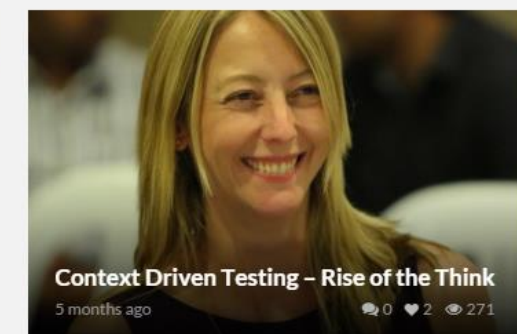
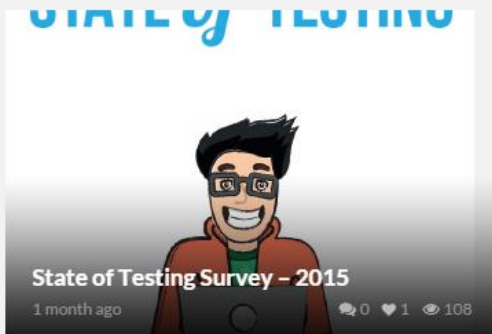
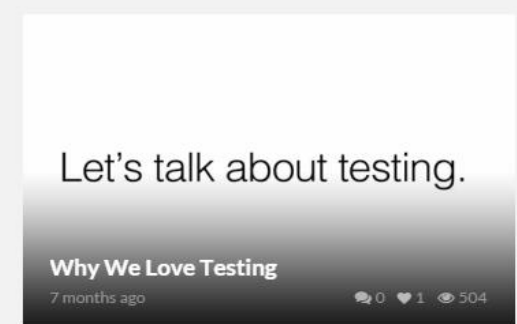
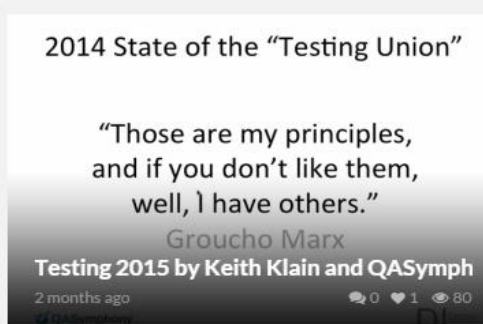
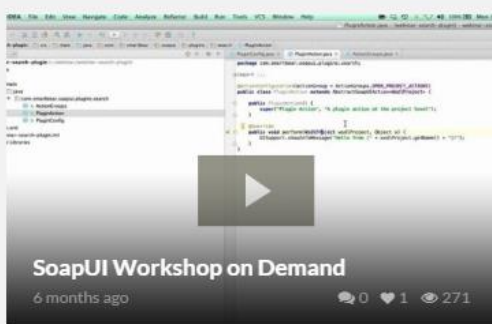
He also imparts short training and consulting sessions, and is one of the chief editors of ThinkTesting - a Hebrew Testing Magazine.

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


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

Joel Montvelisky

Georgios Kogketsof

...and others

our family

Founder & Editor:

Lalitkumar Bhamare (Pune, India)

Pratikkumar Patel (Mumbai, India)



Lalitkumar



Pratikkumar

Contribution and Guidance:

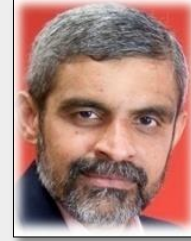
Jerry Weinberg (U.S.A.)

T Ashok (India)

Joel Montvelisky (Israel)



Jerry



T Ashok



Joel

Editorial | Magazine Design | Logo Design | Web Design:

Lalitkumar Bhamare

Cover page image – marimerveille.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 our group at



Join our community on



Follow us on - @TtimewidTesters



www.teatimewithtesters.com

