THE TESTING PLANET

July 2012 | www.thetestingplanet.com | No: 8 £5

Think you're funny, do you?

WIN!

Show us what you're made of by entering our cartoon caption competition - find out more on page 10

Your Software Test Team could be Just like This Guy! Or something. It's a metaphor, get it?

Leaning up software testing

By Matthew Heusser

I know what you are thinking. "Oh look, an article on Lean. Yes, yet another concept borrowed from an entirely different industry, from a different time in history, that has nothing to do with Software Development. Joy." That statement is best read as intended, dripping with sarcasm.

Wait! Stifle that yawn! Don't flip that page yet! If you haven't heard of lean, sooner or later, it is likely that you will. Some manager, consultant, or expert is going to ask you how your ideas line up with lean thinking.

Wouldn't it be nice to have an answer? A serious answer. A sincere answer. The folks at The Testing Planet

Continued on page 2

ALSO IN THE NEWS

LEARNING FROM CYBERNETICS
The word 'cybernetics' comes from the Greek word Κυβερνήτης...
Continued on page 6

HOW LEAN IS YOUR TESTING?
Lean software development is gaining support but how does...
Continued on page 16

CONTINUOUS INTEGRATION
If you want to be lean, get to the point. Software quality is often as much...
Continued on page 22

THE EVIL TESTER QUESTION TIME
More provocative advice for testers who don't know what to do!
Continued on page 24

The future of Software Testing (Part 3) The Cloud

By Seth Eliot

The future of testing cannot be covered in a mere three parts. But a look at software and the industry indicates that the three areas covered in this series may be a good first place to start. To review:

- Part 1 (The Testing Planet, October 2011) - Testing in Production (TiP) - Leveraging real users and live environments to test software.
- Part 2 (The Testing Planet, March 2012) - TestOps - Live site focus for testers, merging aspects of the Development, Operations, and Testing roles.

Now we arrive at Part 3 - The Cloud.

The Cloud - A Very Brief Introduction

"The cloud is a self-service on demand way of accessing computation resources with a virtualized abstraction.... it's a public service... you can treat it as a utility" - Ray Ozzie, former Chief Software Architect, Microsoft[1]

It's a utility. You pay for what you use. When you are using nothing, you pay nothing. This is "the power

Continued on page 4

LETTER FROM THE EDITOR

Welcome to the Lean issue! I wrote a blog-post on The Testing Planet site a couple of months ago now, to the effect that I was inspired by the lean principles we would be majoring on in the July Testing Planet. That's still the case, but I've realised it's not enough to just be inspired. Lean thinking needs to become a part of my DNA; a catalyst for improvements in the design and delivery of my test projects day-in and day-out.

These are austere times we're living in. Your organisation is probably already giving some serious thought in regards to where and how project lifecycles can be delivered faster with less waste. Testing has historically been something of a bottleneck, but improvements to test processes are, to a greater or lesser degree, directly under your control.

Is your manual testing finely honed like a surgical scalpel, or more like a rusty saw? Are you hooked right into the full development lifecycle of your project, identifying problems and recommending solutions from the product's inception? Is your test design and execution just-in-time, able to respond and adapt quickly to change, or are you spending days or weeks refactoring scripts every time product requirements change?

For many of us, it's time to take a step back and consider where and how lean principles can be applied. You'll find some exceptional thinking on the following pages. Just make sure you implement some of them!

Simon Knight

THE TEAM

Rosie Sherry
Thomas Harvey
Stephen Hill
Mike Talks
David Greenlees
Pete Walen
Catherine Powell

Main story continued from page 1

thought it was a reasonable request.

The result of all this work is the article in your hands. My goal is to inform and entertain, but mostly, I want to honour your investment of time. So let's get on with it and talk about lean testing, shall we?

The first word: Lean

The term 'Lean' was first used by two Americans, James Womack and Daniel Jones, in their book The Machine That Changed the World. In the book they link together a series of innovations in process and thinking that the Japanese used to, well, eat the lunch of American Car Companies.

It's a good story; it is a compelling story. It is tempting to over-simplify the story into Lean=Eating America's Lunch and "succeed by following the rules in this checklist". The real story is a bit more complex.

Let's start with where the myth went wrong - that Lean has its roots in American history, because Henry Ford invented the assembly line.

The Assembly line was a major innovation for its day, and the Japanese did, indeed, visit, take notes, go home, and make very productive factories. But that's not what I mean when I say lean - it is not a series of techniques!

As the founders of lean continued to visit American plants over a series of years[1], they saw one thing in particular: American plants had problems, and they were not being fixed. This is likely because of the American focus on stable, predictable and repeatable plant operations. If you standardise on it, yes, you can experiment and measure results, sure, but you have to actually try different things! Organisations that focus exclusively on standardisation, well, are not trying different things.

If I had to pick a first pillar of lean to stand on, it would be just that: Continuous Improvement. That's a tricky word, continuous improvement. Changes on software projects are more like tradeoffs, where to get something; you have to give something up, aren't they? Or, as Eric Sink, the CEO of SourceGear Corporation once put it "You can't eliminate problems, but you can make trades to get the problems that you prefer over the ones you have now."

The rest of the article is going to try to help you figure out what improvement means for you.

To Get Lean, Drive out Waste

When the Japanese were looking at how to design their factories, they saw one big area that could be improved: Waste. Real waste; actual physical piles of parts that were rejected that had to be thrown away - metal that was bought, paid for, put through a process and then hauled away by a trucking firm. (That charged an additional fee). Beyond the scrap, they also saw time spent on fixing broken parts ("re-work"). This fixing meant the company paid labour to build two parts, but only got one.

In addition to the scrap and re-work, there was waste in employee time. Americans were using a kind of accounting called cost accounting that counted things like the cost to produce a part based on the hourly rate of the man driving the machine. This meant that if you lined up 1,000 parts in a press and the batch only took a few minutes to run, your price per part was extremely low. It also totally fails to consider setup and tear down time, and the time the workers are sitting idle at the next step in the process, waiting for those 1,000 parts to be lined up and fired. We're out of space here, but Eli Goldratt's "The Goal" has an excellent, detailed explanation of this problem. The subtitle of "The Goal" is "A process of continuous improvement." You'll like it.

The Japanese term for this waste is Muda, a word that loosely translates to ... well ... waste. On the Toyota Production System (TPS), a way of mass-producing cars on demand, was designed with this as a primary thought: Continually Improve by Driving out Waste.

The How of Lean Manufacturing

Implementing lean manufacturing is hard to explain - doubly hard in one article. Triply hard when I get about a third of the article to explain the concept - but here goes the short version.

When the folks at Toyota wanted to improve performance and drive out waste, they looked at the problem domain, and designed a system that would produce automobiles at the rate of demand. This is very important; the system they chose fitted the problem. Most of us are not building automobiles, and production is a solved problem; it is a Copy/Paste or File > Save As. Our issue is not production, but we can look at those ideas for inspiration. This point bears repeating.

I belong to something called the Context-Driven School of Software Testing, which goes so far as to claim that There are No Best Practices; that practices are instead better or worse in a given context. The Toyota Production System (TPS) solves a problem, but you may not have that problem. And, as Mr. Sink pointed out so well, adopting one solution may create problems bigger than what it solves!

Now the Americans who took TPS and turned it into Lean did create a set of techniques, and these make sense, and many apply in a software organization. Allow me to 'hit the highlights':

The Present State ("Before"): Imagine having a plant that did all of its work at each stage, then moved all the work to the next, then the next, until the parts go out the door. While one action is taking place somewhere in the plant, everyone is idle. This is bad. Ironically, it also sounds a lot like the waterfall development process.

The Future State ("After"): The opposite of this stops-and-jerks big-batch-size approach is smaller batches, then smaller still, until you have one-piece flow, in which each individual piece flows throughout a system. This means the folks next in line can get a part as soon as possible. The basic goal of TPS is to achieve flow, and to do it by eliminating these seven muda:

- Transport (moving products that are not actually required to perform the processing)

Continued on page 3

Real Examples of Bad Bug Reports - http://bit.ly/badbugsreports

Continued from page 2

- Inventory (all components, work in process[2] and finished product not being processed)
- Motion (people or equipment moving or walking more than is required to perform the processing)
- Waiting (waiting for the next production step)
- Overproduction (production ahead of demand)
- Over Processing (resulting from poor tool or product design creating activity)
- Defects (the effort involved in inspecting for and fixing defects)[3]

Lean in the IT Shop

Things are wildly different in a software organization, and it is easy to get horribly confused. For example, using "motion waste" as an excuse to standardize where the stapler goes on every desk, or force a "clean desk" policy where no papers can remain anywhere, that is just ... sad. That might have something to do with production on an assembly line, but in software, we have more important things to focus on.

When I say lean, that is not what I am talking about. I do, however, see motion waste in poor tooling, in waiting two days to provision a test machine, in having to create a ticket in one system, then cut/paste data from another, then hand key data from a third into a fourth just to get the permissions changed on a directory so you can run a process. That kind of stuff is waste, it happens all the time, and it won't change without concerted effort.

Likewise, some of the defects found by testers are waste. I say some because the companies I have worked with that tried to eliminate defects – to prove the software correct in one way or another - spent a huge amount of time, energy and effort and still had bugs. That kind of analysis paralysis is its own waste. Here's part of why: While Cleanroom methods[4] have been studied and work in certain cases, they tend to work only if the company owns the entire solution, down to the hardware. In an era when software developers program on one browser and we have to support twenty combinations of browsers and versions, each with a different JavaScript interpreter, it is unlikely that we will prevent all the defects.

While we might not prevent all defects, all of the groups I have consulted with had opportunity to improve the quality of the code before it gets to test. That means less time spent reproducing, documenting, triaging, and re-testing. It means the code can be deployed to test more quickly.

Overproduction is another waste. Consider, for example, the company that has five analysts, who have 'analyzed' a year worth of projects, and are 'just waiting' for the technical team to go and code up the existing work.

Imagine those business requirement documents not as files on a computer, but physical sheets of paper in an inbox. That is a lot of work in progress inventory.

Meanwhile, what are the analysts doing? Working on the project your team might work on twelve months from now.

Twelve months from now the company priorities may not be the same. The technology landscape will be the same; the team organisation may change! We can start to see those new business requirements as waste; the analysts would be better off figuring how they could help test and deploy the existing applications. That would actually help improve time to market for the next three years!

Seeing overproduction as waste moves us toward just-in-time analysis. That means that as soon as I, the tester, am overloaded, the programmer can no longer push work onto me. If he finishes his task, he needs to help me get my existing tasks finished before assigning the next one. That might mean creating automation tools, doing some test automation, helping out with the hands-on test effort, or running to the sandwich shop to buy lunch. The point is to move from a push system to a pull one, and the same rules apply for the developer as do for the tester.

The second word: Testing

My description above focused on Software Development, and that was on purpose; Lean is a whole-process concept, that focuses on the entire value delivery chain, what Mary Poppendieck refers to as "concept to cash" in the book she co-wrote with her husband, Tom, which has the non-ironic title "Implementing Lean Software Development: From Concept to Cash."

Looking at Software from a Lean perspective, we want to optimize throughput. That means finding the bottlenecks and escalating them; making them important and adding helps where possible. If the bottleneck isn't testing, but is instead development, well, we can focus on that first, maybe working on handoffs from developer to tester. In this way, the term "Lean Software Test" doesn't make a huge amount of sense. And yet...

The teams I have worked with that implemented the "push to pull" approach I described above found some specific issues for testing. Here are the four I recognize most readily:

Regression Testing - You might want to deploy every minimum marketable feature as soon as it is tested, in order to achieve one-piece flow. Yet that feature is a part of a greater system, and your new, tested feature might introduce a defect somewhere else. So you want to regression test the system, and trying to regression test the entire application for each feature introduces a big, huge bottleneck called "testing."

Blowback - Everything goes great until a tester finds a bug. Then the developers, who have a work-in-progress limit set, suddenly have an extra thing to work on. Switching from that extra thing back to the story they thought was done causes disruption; now the folks that wanted to work on fleshing out the next story can't because they cannot 'push' the current story to dev. Meanwhile, the testers are blocked, waiting for a build. As an occasional thing, this happens (it sure happens in traditional software development!) but a lot of blowback can cause thrashing, which is a huge danger to productivity.

Over Thinking the Factory Analogy - Software is not a factory. Even if it were, modern factory theory is nothing like the clichés of a General Motors factory in the 1920's that many people think of when they hear the term. Over-thinking the factory analogy can lead to treating humans like cogs in some sort of machine and sometimes the notion that testing is unskilled manual labour and best (entirely) automated. I am all for test automation, don't get me wrong, but what computers do when they run through pre-recorded steps in a specific way, with pre-defined inputs along a specific path leading to certain expected outputs -- that is not what a good tester does. Trying to automate that tends to lead to comprehensive, but brittle tests that were very expensive to create and maintain, or incomplete automation that allows bugs slip through the net: sometimes both.

Waste in Test Ideas - Some tests take a long time to run, always return success, and test a feature that, if broken, would not be a showstopper. If they never ever give an error, there is a case to be made that they are muda -- but that is sort of like saying a car insurance policy is muda, isn't it? Deciding what tests can be skipped, when, why and which tests must run, is a challenge for any test organization, but it becomes more visible when the organization has an aggressive "drive out waste" policy.

The two words together: Lean (Software) Testing

Most organizations that I talk to that are adopting a "Lean Software" approach take that approach to the entire technical team. Testers are usually embedded in the technical staff, reducing the time it takes to walk over to a developer or product owner and ask

Continued on page 4

AUTHOR PROFILE - MATTHEW HEUSSER

This article may have been written slightly tongue-in-cheek, but Matthew Heusser is a no joke. A consulting software tester and self-described software process naturalist, who tests, manages, and develops on software projects. Lately Matt has been coaching testers while contributing in short bursts - a sort of "Boutique Test Consulting" that actually leads to quicker time to market right now, on this project right here. A contributing editor for Software Test & Quality Assurance Magazine, Matt blogs at "Creative Chaos" sits on the Board of Directors of the Association for Software Testing, and recently served as lead editor for "How to Reduce the Cost of Software Testing" (Taylor and Francis, 2011). You can follow Matt on twitter @mheusser or email him matt@xndev.com or read more on his blog: http://xndev.com/creative-chaos

Some great advice and discussion about learning Selenium - http://bit.ly/seleniumcourses

Continued from page 3

a question. The organizations tend to remove layers of management and bureaucracy, because time spent routing documents and information is its own form of muda. Testers think of themselves as part of a greater product team, with a role of helping to deliver software. Not just software, to software with a purpose, or a value: Enabling a business process, solving a problem, making something of economic value that people are willing to pay money for.

Most of the teams I have worked with drop 'iterations', limit work in progress to the number of dev-pairs available at one time, encourage pair programming, collaborating with testers, focus on Mentions-In-Passing over formalized bug reports, and measure cycle time (days it takes a story to move from active work to deployed) and throughput (number of stories accomplished per week.) The metrics the team uses are whole-team metrics.

The teams also have to deal with a very compressed regression-test window. They usually expect testers to improve quality before the code gets to test, to minimize blowback. Often the developers automate examples they are given before coding starts; testers work to help create the examples up front. Once the software is ready to 'test', the tester verifies the automated checks are checking the right things, and then perform exploratory testing. In order to tighten that window, testers tend to get very good at exploratory testing and evaluating risk. Companies with a large, integrated set of applications tend to both develop a test cadence, and deploy a small batch of features at a time, or else enable quick deploys, intense system monitoring, and a very big 'rollback' button in production.

Now there are many reasonable interpretations of 'lean'. Most of these teams follow the tradition culled by Womack and Jones from the Japanese Manufacturing Revolution, interpreted by two American Academics, and then interpreted again by the Software Industry. Two most well known camps are probably the work of Mary and Tom Poppendieck[5], and the Lean Systems Society[6], but the real heart of lean is continuous improvement by driving out waste.

You may define improvement differently; you may define waste differently. That's okay. The important thing is to take a systems thinking approach and know what you stand for.

One Final Note

In my time in software, I've seen organizations that build a pile of inventory in front of the technical staff, and then yell at them for being the bottleneck, while other people who could help stand idle, all because this "isn't my job." If anything, lean testing has the potential to explain that there is a better way. In that, I rejoice.

If, after reading this article, you can too -- that you can understand these ideas, articulate them, and even see how they could make a difference in your organization, well, I will count that as victory and exit stage left. The next step is up to you.

Good luck! □

REFERENCES

1. I am speaking of Eiji Toyoda and Taiichi Ohno here, who studied under an American name W. Edwards Deming.
2. http://en.wikipedia.org/wiki/Work_in_process
3. Seven Types of Muda as attributed to "Lean Thinking", Womack and Jones, 2003, from Wikipedia http://en.wikipedia.org/wiki/Lean_manufacturing
4. http://en.wikipedia.org/wiki/Cleanroom_software_engineering
5. Lean Software Development: An Agile Toolkit
6. http://leansystemssociety.org/

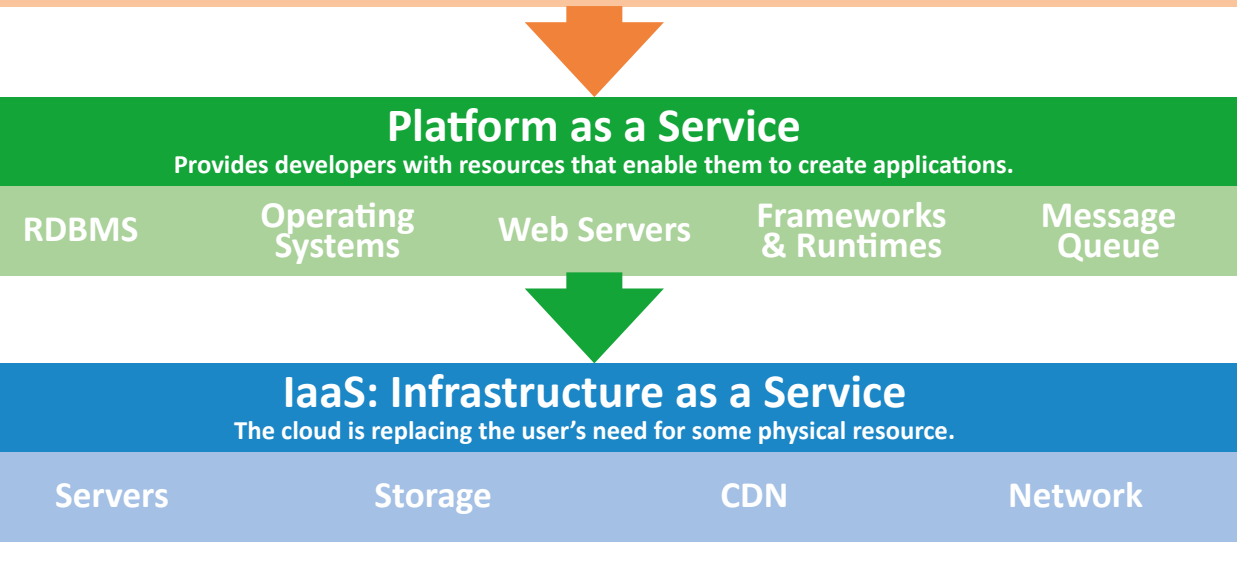
Second story continued from page 1

of zero" to which I will refer back to. This is one side of the coin of the key cloud feature of elasticity - you can spin up as many resources as you need or wind them down to save money when you do not need them.[2]

The other key concept is abstraction. The Cloud abstracts away what you do not care about so you can concentrate on what you do care about.

- When running Hotmail, The Cloud provides Software as a Service (SaaS). The deployment, maintenance, hosting, and even execution of the mail client and server are taken care of for you by Microsoft. They are abstracted away so you can use the functionality to receive, send, and organize mail.
- If you develop and run a service deployed to Amazon EC2, Amazon supplies virtual servers on tap to host and run your service. This virtualized hardware provides Infrastructure as a Service (IaaS). The procurement and racking of servers, as well as the power requirements, air conditioning, and maintenance of the machines are abstracted away so you can concentrate on running your service and serving your users.
- That leaves Platform as a Service (PaaS) straddling the middle. Think of this as software that enables you to develop and run software. For example, if your cloud service provides an RDBMS to store and process data this is PaaS.

Together these three layers of cloud services form a stack as seen in Figure 1.



Three layers of The Cloud form a stack

For industry examples of each of these see Cloud Computing: One Picture and Several Examples.[3]

The Cloud from a Testers Point of View

The Cloud changes the way we test on two levels:

- As in Part 1 (TiP) and Part 2 (TestOps) this is primarily focused on software services. But even when not run from The Cloud (services run conventionally from a data center), there are tools and techniques The Cloud brings to bear on your test approach.
- And if your service is run in The Cloud, then a further level of strategies is opened up to you for use in your test plans. Let's start with these first.

Cloud Enabled Testing for Cloud Deployed Services

Test in Cloud (TiC) - The most straightforward and perhaps most powerful test technique leverages

Continued on page 5

To Gherkin or not to Gherkin? http://bit.ly/togherkin

Continued from page 4

cloud elasticity and the power of zero. If the System Under Test is deployed using the The Cloud, then spin up your test environment in The Cloud also and make it look exactly like your production one. Some call this Test in Cloud (TiC).

When promoting Testing in Production in Part 1, our motivation was to test in an environment as close to (or actually in) the production environment. Because you can provision cloud resources as needed, you can create a test environment that looks just like production. Cost then is the main reason we do not buy thousands of servers to mimic our large scale production systems, but with The Cloud you pay only for what you use and pay nothing when you do not need it. This is unlike a physical lab that requires care and maintenance all the time. Also unlike a physical lab you will always have the same machines (virtual machines) as production; there is no need to upgrade hardware in the lab as production SKUs are updated.

The test deployment differs from production in one key way - it is not carrying real user traffic. Although it requires some more engineering, you may copy real user traffic and send it to the test environment. Real users only see responses from the production environment while engineers can evaluate how the test environment handles the copied data from real usage. This methodology is sometimes referred to as Shadow Deployment.

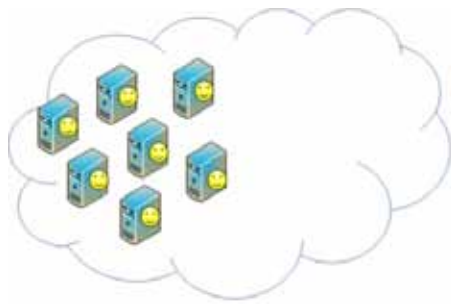
Canary Deployment

Netflix provides instant video streaming throughout the Americas and parts of Europe. Their scale is huge, accounting for 32.7% of internet download bandwidth in North America;[4] therefore they chose Amazon web services to host their service. While ramped Deployment is a powerful and fundamental technique of Testing in Production not specific to cloud services, how Netflix does it with their Canary Deployment[5] would be cost prohibitive for services not in The Cloud. Netflix video streaming is deployed to the Amazon EC2 cloud service. Their canary deployment process is described in Figure 2.

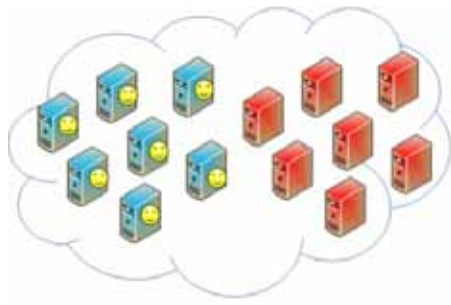
This way Netflix leverages their 1 Billion API requests per day to derive quantitative assessment of the quality of new releases. Problems such as memory leaks which can be difficult to find are easily revealed this way.[6]

Cloud Enabled Testing for All Services

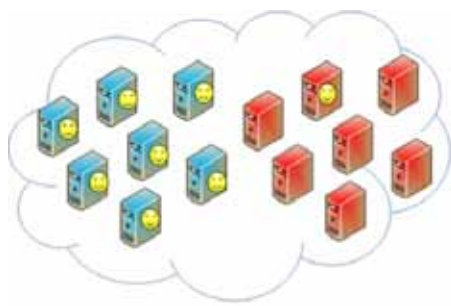
Test in Cloud (TiC)... Again - Even if your service is not deployed to The Cloud, you can still take advantage of cloud technologies to test it. Test in Cloud was a good choice for cloud deployed services as it enabled an instant duplicate of production for use in testing. The challenge for a traditional service is to duplicate the server configurations, network topologies, and security configuration in The Cloud that looks like the physically deployed service in the data center. Amazon Virtual Private Cloud (VPC)[7] overcomes much of this challenge, allowing configuration of routing tables, IP address ranges, and security policies. Using access control lists you can ensure no one else can access your



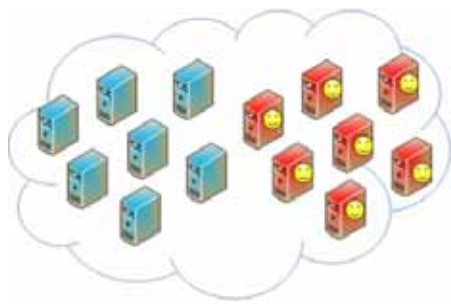
The current version (Vcurr) of a Netflix service is deployed to the cloud and carries user traffic



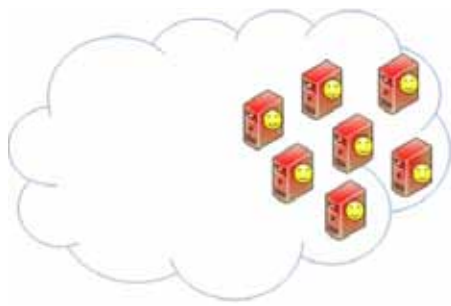
The new version to be evaluated (Vnext) is deployed to the cloud, but carries no user traffic



A single server from Vnext starts taking user traffic. This is the "canary". This server is monitored. If there is a problem all user traffic is easily routed back to Vcurr. If no problems are observed and all goes well then...



All user traffic us moved from Vcurr to Vnext. The Vcurr server instances remain while the new Vnext is monitored for sufficient period to assure engineers that all is well. If a problem occurs with Vnext, rollback to Vcurr is easy



Finally once Vnext has proved itself trouble free, the Vcurr server instances can be de-provisioned

The Netflix "Canary" Deployment in The Cloud

AUTHOR PROFILE - SETH ELIOT

Seth Eliot is Senior Knowledge Engineer for Microsoft Test Excellence focusing on driving best practices for services and cloud development and testing across the company. He previously was Senior Test Manager, most recently for the team solving exabyte storage and data processing challenges for Bing, and before that enabling developers to innovate by testing new ideas quickly with users "in production" with the Microsoft Experimentation Platform (http://exp-platform.com). Testing in Production (TiP), software processes, cloud computing, and other topics are ruminated upon at Seth's blog at http://bit.ly/seth_qa and on Twitter (@setheliot). Prior to Microsoft, Seth applied his experience at delivering high quality software services at Amazon.com where he led the Digital QA team to release Amazon MP3 download, Amazon Instant Video Streaming, and Kindle Services.

cloud based sandbox but you and your test suite.

One Million Users - Another way to leverage The Cloud is for synthetic load generation. The benefits here are elasticity, which lets you ramp up to usage levels to challenge even the most scalable of services, and geographic distribution which enables you to hit your service from all over the globe. The latter is important as your service will behave differently when all load comes from a single source versus geo-distributed load.

You can design your own load generation system or use one already in the marketplace. SOASTA is one such system. In one of their most impressive case studies they generated the load of one million concurrent users on top of actual live traffic to test MySpace streaming music.[8] This test pummeled MySpace with 6 Gigabits and 77,000 hits per second. SOASTA and other such systems like LoadStorm run on top of existing cloud services like Microsoft Azure to achieve high traffic volume and geographic diversity.

Facebook also enables app developers on their platform to create test users. The goal here is not so much scale as data integrity. Facebook test users can neither see nor be seen by real users. Without such a capability, app developers would create free real users, thus clogging up Facebook with bogus data. Facebook also gives developers the ability to create and destroy these test users programmatically via an API, enabling test automation. In this case Facebook is leveraging their cloud platform to provide test users as a service.

Big Data - As discussed in Part 2 TestOps, testers will become more interested in leveraging the big data pipe (large telemetry stream) coming from their services for use in quality assessment. To store and process this big data we turn once again to The Cloud. While Hadoop is a popular open source system for just such a task, one still

Continued on page 6

What would a testing museum look like? http://bit.ly/testingmuseum

Continued from page 5

must procure the hardware to run it then deploy and maintain a Hadoop cluster. Instead you can abstract away these tasks by using Microsoft Azure's new solution which provides the Hadoop platform as a service (PaaS).[9] Alternatively Google provides turn-key data analysis of terabytes of data with its own BigQuery solution.[10]

Ignore at Your Own Peril - The advantages of The Cloud for service development and deployment will continue to drive cloud[11] growth. Microsoft devotes 90% of its R&D budget to cloud , while Amazon continues to see amazing growth in its cloud services, doubling the cloud storage by users in just 9 months.[12] Testers need to be aware of The Cloud and how it can be used to deploy and test high quality services. □

REFERENCES

1. http://techcrunch.com/2009/06/04/liveblogging-microsofts-ray-ozzie-on-the-potential-of-cloud-computing/
2. http://blogs.forrester.com/james_staten/10-05-20-could_cloud_computing_get_any_more_confusing
3. http://blogs.msdn.com/b/seliot/archive/2011/07/25/cloud-computing-one-picture-and-several-examples.aspx
4. http://www.technolog.msnbc.msn.com/technology/technolog/netflix-uses-32-7-percent-internet-bandwidth-119517
5. http://perfcap.blogspot.com/2012/03/ops-devops-and-noops-at-netflix.html
6. Slides: http://www.slideshare.net/joesondow/building-cloudtoolsfornetflix-9419504
   Talk: http://blip.tv/silicon-valley-cloud-computing-group/building-cloud-tools-for-netflix-5754984
7. http://aws.amazon.com/vpc
8. http://highscalability.com/blog/2010/3/4/how-myspace-tested-their-live-site-with-1-million-concurrent.html
9. https://www.hadooponazure.com/
10. https://developers.google.com/bigquery/
11. http://www.forbes.com/sites/kevinjackson/2011/04/19/cloud-to-command-90-of-microsofts-rd-budget
12. http://aws.typepad.com/aws/2011/10/amazon-s3-566-billion-objects-370000-requestssecond-and-hiring.html

What testers can learn from Cybernetics (of Cybernetics)

By Stefan Kläner

The word 'cybernetics' comes from the Greek word Κυβερνήτης; the act of steering. So what does the captain of a ship do to safely maneuver his ship into the harbor? He is constantly adapting the process. If the ship drifts to the left because of wind conditions, he estimates the deviation and applies countermeasures to correct the error. The result is a possible course deviation to the right, so he has to estimate the deviation towards his goal (Τελος) in each moment. What is happening here? The steering (the cause), produces an effect; the course deviation. And then the effect becomes a cause - another course deviation. This example introduces circular causality and during the article we will experience why circularity is the essential principle of cybernetic thinking.

Cybernetics

Cybernetics arises when an effector (e.g. a heater, an engine, etc.) is connected to a sensor mechanism, which acts with a signal upon the effector. But cybernetics, much like testing, has many different definitions:[1]

- **Norbert Wiener** - The study of control and communication in the animal and the machine.
- **Stafford Beer** - The science of effective organization.
- **Gregory Bateson** - A branch of mathematics dealing with problems of control, recursiveness and information.
- **Gordon Pask** - The science of defensible metaphors.

- **American Society for Cybernetics** - The study of systems and processes that interact with themselves and produce themselves from themselves.

Norbert Wiener, who re-introduced cybernetics into scientific discourse in 1948, observed "the behavior of such systems may be interpreted as directed toward the attainment of a goal".[2] What he means is that systems have a goal, and they use information to get to the goal. The behavior that Wiener observed was later called feedback. Human beings that interact with their environments have goals, and they use feedback to get to their goals, so first-order cybernetics is a science of feedback (circularity), information and goals.

Wiener founded cybernetics on the metaphor of mechanism but many confused

Continued on page 7

Get your one page guide / mindmap to mobile testing - http://bit.ly/startingmtesting

Continued from page 6

mechanism with machine, and treated both as if the controller, the feedback generator, was separate and external from the system. This may sound familiar. As testers we act as feedback providers, and for a long time we were not seen as part of the system. Indeed, there are still many who support the idea of an independent, external test team.

First-order cybernetics follows the scientific principle of objectivity and separates the subject from the object and refers to an independent world. Wiener ii stated that control is linear and causal: the controller controls the controlled. That would indicate that the meaning lies in the text and not the reader. Furthermore, any responsibility is passed on to a higher level of hierarchy.

Second-Order Cybernetics

In around 1970 second-order cybernetics came into being. It is characterized by its circularity and by the paradigm of inclusion, which means the inclusion of the actor/observer. Heinz von Foerster[3] talked about "cybernetics of observing" as opposed to "observed systems" (first-order).

So while first-order cybernetics talks about linear causality and the strict separation of the observer and the observed, second-order cybernetics adapted the logic of George Spencer-Brown.[4] He wrote in his book Laws of Form: "Draw a distinction and a universe comes into being." In his opinion the act of distinction is the most essential operation of thinking. If we want to describe something we have observed we must start by drawing a distinction, because if we have not chosen a distinction first then we are not able to describe anything.

Second-order cybernetics violates[i] the basic principle of scientific work, which is the principle of objectivity. The properties shall not enter the description of his observations or in other words, there must be a strict separation between the observer and the observed. Heinz von Foerster[5] describes why this violation is necessary: "If the properties of the observer (namely to observe and describe) are eliminated, there is nothing left; no observation, no description." He also encourages the observer "to speak about oneself," therefore shifting the way of looking at things from "out there" to looking at "looking itself."[ii] So he claims that every cybernetician is responsible for his action/observation and manifests it by saying "a cybernetician, by entering his own domain, has to account for his or her own activity. Cybernetics then becomes cybernetics of cybernetics".[vi]

First-order cybernetics uses specific concepts, assumptions and theories that are not reflected; so you basically just act. Second-order cybernetics asks questions as: "What is the purpose of the purpose? What is the goal of the goal?" Nothing is taken for granted.

Through reflection about the purpose and the goal of observing, second-order cybernetics introduced the notion of the responsibility of an observer for his observing. Glanville[6] goes a step further and says the observer is not only responsible for his observing but also for "its frozen version, which we like to call observation. He is responsible, it is his, his own, he owns it and he must own it — as the therapist will tell us. We, as humans, as cognitive beings, must take responsibility for our observing (our knowing, our living, our acting, our being . . .) for we cannot pass on our observing: it is ours, integrally ours."

Probably the best insight of cybernetics is that all sorts of things go wrong: uncertainties, limitations, understanding, descriptions or everything else we have not imagined or thought of.

Sociology in particular has adapted the notion of circular causality, under the influence of Luhmann's social systems theory[7] and later Baecker[8]. But the adoption of circular causality can be found almost everywhere in social sciences. Bateson[9] used it to describe his idea of double blind. Glanville[6] and Pask[10] discovered that design works according to circular causality. Beer[11] also considers it an essential concept in management cybernetics.

So we have seen that circularity is an essential concept of both first-order and second-order cybernetics. But although we have to deal with non-linear systems, we still believe in linear cause and effect. If cause and effect are indeed linear, where do we stop? A influences B, B influences C, C influences D... Until we are on a level of molecules and atoms. We just have to accept that everybody influences everybody and everything is interconnected.

What Cybernetics Teaches Us

Ashby's Law of requisite variety[12] follows the thought that every system has boundaries. For example, if you want to go to X but you are off the path by 10%, you correct your course and you are fine, but when you are 25% off you are screwed. Imagine a program with a number of critical defects, it is probably not a nice situation but you can fix those defects and keep developing the software. But if the software is so screwed up that it does not make any sense any more to fix all the defects, it is time to cancel the project. Or imagine an air conditioning system: When the temperature is 30 degrees it will work fine and bring it down to 18 degrees, but if the outside temperature reaches 35 degrees[iii] it might fail in bringing the room to a comfortable temperature. So the system (the room, the thermostat, the air conditioning, and so on) has a certain variety. Requisite variety is the ability to achieve the goals that we have for it. Classic examples are biological systems, e.g. human beings. There has to be requisite variety to keep the body temperature at a steady level, to keep the glucose level at a steady level, etc. Ashby[xii] refers to those as essential variables and the human body has the requisite variety to keep the essential variables steady in order to stay alive within certain limits. Bringing us back to software, you can develop a piece of software to do whatever you want, but the more variety it has (more features, more capacity, etc) the harder it is to build. So the amount of variety you want to program into the software has trade-offs. Awareness of the trade-offs (the limitations of the system) is critically important.

Heinz von Foerster connected his idea of the ethical imperative with cybernetics. Which he defines as follows: "Act always so as to increase the number of choices."[13] What Heinz von Foerster means is that we should not limit the activities of someone else and instead we should act in a way that enables the freedom of others and of the community. The higher the level of freedom, the more elevated the level of choice and the greater the opportunity to take responsibility for our own actions. Conversely if we limit someone's freedom we take from him the chance to act responsibly. Of course it is easier to hide behind a hierarchy and to say "But I just followed orders! It was not my fault! There was no other choice!" My point is, we should not limit any co-workers freedom. We might not like opinions because they do not match with ours, but we should offer alternatives[iv] instead of judging.

Let us take a look at the certification debate: Instead of categorizing testers into certified and non-certified groups we should offer alternatives. It would be healthier for our craft if we stop judging people for taking part in a certification course or for being certified. Do we know what is the best, the right, the bad or the wrong? There is no single or absolute truth that forces someone to see things in only one way and to act in only one way; we are free to choose, free to decide. Wittgenstein[14] taught us that the truth is tautological. Because in the end, to extend Baecker's "if we turn down the belief that the world is full of things, attributes and characteristics and instead, believe that it is full of processes whose beginning is unknown, we are faced with the unbelievable phenomenon that a frog is a frog, love is love and money is money"[15], a test is a test.

Due to the self-referential nature of cybernetics, von Foerster[v] proposes a shift from "Thou shalt [not]..." to "I shall [not]..." because "we can only tell ourselves how to think and act". So self-reference teaches us that A to B, B to C, C to A = A to A. Pradeep Soundararajan tweeted a couple of months ago, that "Testers don't improve quality, those who listen to them do".[v] My reply was that I disagree with him, and that testers only contribute to a causal chain (circularity). But due to the limitation of 140 characters I never explained why; so let me do it now. A tester influences a programmer, a programmer influences a project manager and the project manager influences the tester. So the tester influences himself. That is, why, in my opinion, Pradeep's tweet should have been: "Testers, who listen to themselves, improve quality." Because the tester by the way he acts, influences the whole team and vice versa.

Second-order cybernetics is a study in which observers and actors take responsibility for their observations and actions and as a result it

Continued on page 8

AUTHOR PROFILE - STEFAN KLÄNER

Stefan Kläner works in the software testing field since 2008 with a main focus on usability and acceptance testing. Stefan is a firm believer in self education. When he is not testing he researches on systems theory, design thinking, sociology and psychology. He also occasionally tweets as @sklaener.

Can we create a mobile test lab? http://bit.ly/mobiletestlab

Continued from page 7

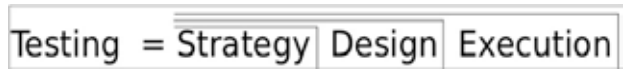
encourages us to accept errors. No matter if they are intentional, out of ignorance, by opinion or otherwise created. All errors, failures, mistakes, etc. are welcomed and remain the responsibility of their owner.

Any observation needs an observer; cybernetics of cybernetics insists that there are processes, and that we are involved with and in our processes. It insists that there is no observation without an observer[iii], no knowing without a knower, no communicating without a communicator,[vi] [16] no thinking without a thinker, no thinking without thinking, no testing without a tester and also no testing without testing. Having said that, the creation of an automatic test-script insists of a tester, and is therefore allowed to be called testing. The pure execution of an automatic test-script is not, because testing requires a tester, not a machine. So cybernetics of cybernetics supports the distinction Michael Bolton[17] [18] has drawn - to say it with Spencer-Browniv - between testing and checking. Some people argue, that one day these tools might be intelligent enough and that we then have to acknowledge that the tool is indeed testing and not checking. Alan Turing was confronted with similar questions: Can machines think? And does artificial intelligence exist? In order to find an answer, he came up with the Turing-Test[19]. The Turing-Test, which is still used in the science of artificial intelligence, consists of an entity behind a curtain, and it does not exist further information about the entity. Now several scientists ask the entity questions, and after a while of questioning the entity, they will judge if the entity is a human-being, a machine or that it is not decidable. If the scientists interpret the given answers as one from a human-being but, instead, the entity is a machine, we have to acknowledge that the machine is indeed an intelligent one. Several cyberneticians addressed this test, and Heinz von Foerster[13] asked the question: "Do we really verify the possible intelligence of the machine, or do those scientists testing themselves?" They are indeed testing if they are able to differentiate a machine from a human-being or vice versa. Heinz von Foerster[13] framed his conclusion in the wonderful way: "Tests test tests".

Another problem of test automation is the fact that the tools, of course, contain bugs as well. But one of our desires of automation is reliability, which a tool can certainly not fulfill. Or as Glanville[20] frames it: "It is the irony of our image of the machine that we take the machine-metaphor to indicate unfailing, predictable reliability, whereas all we can really predict about machines is that they fail."

Cybernetics & Agile

Agile Software Development is full of circularity; actually it is circular in itself. But most approaches to Software Testing have almost no circular notions. We are trying to involve the customer in the process, but this mostly helps on the level of ATDD or, as Gojko Adzic calls it, Specification by Example. If we use the Laws of Form by Spencer-Brown[iv] to illustrate a traditional approach to Software Testing we would end up with something like this:



Allow me to re-frame an interpretation of Ashby's Law of Requisite Variety: The circularity of a verification and validation system must be equal to or greater than the circularity of the system it tries to verify and/or validate. Glanville[21] extended Ashby's Law, and said that it, in fact, needs exactly the same variety. What do I mean by this? Only if we increase the feedback, which we provide ourselves, can we handle the problems we are facing. Change is all around us and while we claim to respond to it, our traditional approaches fail miserably. We are either not fast enough or do not find the important defects, or even both. I admit, it is unfair to say that our traditional approaches have no circularity at all, we do reflect on our test design and strategy, but usually not systematically enough.

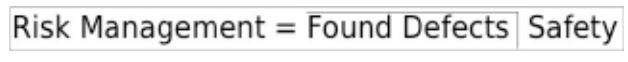
Exploratory Testing is definitely a step into the right direction, but does our entire testing have to be exploratory? Certainly not, but Exploratory Testing seems to work very well for a lot of people in Agile environments. My theory is that the reason for it is the circular character of Exploratory Testing and that is why I believe it is beneficial to adopt the form of Exploratory Testing. Bach[22] defines Exploratory Testing as "simultaneous learning, test design, and test execution". Let me model this as well, because "when the present has ceased to make sense, it can still come to sense again through realization of its form"[iv]:

Now we can see how circular it is, we can expand it further and see even more circular aspects. There is no need for it to happen simultaneously; it is just important that the reflection about what has been done goes back into the design process or into further test ideas, and therefore further test charters. That is what Spencer-Brown[iv] and Luhmann[vii] call re-entry. If we have no feedback in our own process, how can we get better giving feedback to others? There is also no need to call it Exploratory Testing, or anything else since the name is really not important. An argument uttered twice does not make it more true, a test executed twice (without a code change) does not make it more likely to find defects, or to say it with Spencer-Brown[iv] "to recall is to call". There is more than one way to react on the change around us. But the chosen way has to be viable. It needs to fit in the context; it needs to be compatible with the system. We need to accept and understand the fact that in evolutionary processes, it might appear from the outside view, that we do not choose the best solutions but some solution. But to assess those as better or worse is an observation of second-order and therefore only questionable from the inside.

Our process will become unmanageable when it is not possible for us to provide the necessary feedback in time. At first, unmanageable may sound like a problem, but it might actually be a desired state to be in. Agile already has the notion of self-management, so why do we still empower Test-Managers? Instead of one controlling/observing all the others, all participate in shifting towards group self-control. So every Tester becomes a Manager, because control is "neither action nor reaction: it is interaction"[23]. We need a shift from Testers, which look at their job descriptions and act upon it, to Testers that test with all their senses. They do not only need to get their work done, but they also need to observe what their colleagues are doing. We need to observe if our colleagues get their work, on which we highly depend as a group, done. Our ability of cooperation does, therefore, heavily depend on our ability to observe and not so much on the ability to talk. We need to include, the previously excluded (or the unmarked state as Spencer-Brown calls it), in our observation again.

That is where Pair-Testing or a group approach to Testing (or Testing Dojos) kicks in. These approaches are valuable, but not necessarily suitable. Not suitable because some people prefer to test on their own. And what we are doing then is telling them to get out of their comfort zone. Let us apply the already mentioned ethical imperative, are we really in the situation to tell people how to be most effective? I most certainly doubt that. Yes, I would even claim that they are most effective when they feel most comfortable. If we embrace the state of unmanageability and implement daily reflection sessions (whatever form they may have), in addition to stand-up sessions, we can enhance our creativity by borrowing ideas and knowledge from others. By reflecting as a group, not in small debriefing sessions, we increase "our number of choices", our possibilities. But only if we reflect together on how we tried to reproduce a bug, why we selected the strategy we selected, why we used the techniques we used, etc. Perception already forces us to focus and therefore we will only see part of what is possible to see. That is why two persons, who observe the same event will make different observations. And only qua reflection (and, of course, self-reflection) can they compare their observations and gain more insight. So we need an open mind and to keep listening and observing, otherwise "we stand a good chance of missing possibilities by imposing on it our lack of imagination and blindness."[23]

When I say that we need to reflect more about what we are doing, and why we are doing it, does it mean that we are currently not able to learn? As I see that Ad-hoc Testing is slowly disappearing, which of course is a good thing; we can say that we do learn something in this regard. So my observation is, that we are able to learn in our daily work (using techniques, applying heuristics), but we still struggle with our semantics. There are still many who interpret found bugs as if they would imply a measure of safety.



Therefore we pretend something towards our customers, our management and ourselves, which does not reflect our practical work; seeing found defects as potential dangers, risks.

Continued on page 9

Become a Test Ninja! www.testninjas.com

Continued from page 8

Risk Management = Found Defects Risks

Cybernetics & Quality

Weinberg[24] stated, that "quality is value to some person" and Michael Bolton[25] went a step further and said "X is X to some person". While I value both versions, I want to make a suggestion for a short, modern[vii] notion: Quality is observer-dependent. And, of course, the observer changes his way of observing over time.

Cybernetics is a way of looking at things, a way of understanding what is going on. We should not ask what cybernetics is, but when it is. Imagine a child that tries to get an apple from a tree. And the child takes a stick to get the apple. In this case cybernetics is not the apple it is the stick.

HOW DO WE GET THERE? HOW CAN WE DO THINGS? HOW CAN WE ACHIEVE THINGS?

Information could be anything but it does not make sense if you do not know how to interpret it. So in this article there is data, but only when you read it you generate information. Data is nothing and form is only in the eye of the receiver. Information is generated by the one who looks at things.

Observing is a more general dynamic than Heisenberg's demonstration that if you observe something, you alter it. It is more like: We configure (in terms of design) the room that, while we configure it, configures us. In conclusion: observing is not a passive activity, an import of information. How we see things is not determined by the object alone. If you kick a dog, the reaction of the dog is not determined by the energy effect (caused by the kick) but by its inner structure and past experiences (Bateson[ix]). The same applies to our observations.

We should stop talking about the content of testing and start talking about the form of testing, in which the content is expressed (everything which is expressed, is expressed in a certain form). There are different perspectives and ways to interpret the Laws of Form. One way is to understand and to comprehend processes. Another one is to understand the content, which is formed in and by a process. But it is also important to see that the process (the distinction) and the content, as a result, emerge together. Or to say it with Varela[26]: "Wanderer the road is your footsteps, nothing else; you lay down a path in walking". So the path emerges while walking. If we continue to talk only about the content, and one could argue that we are doing so since Glenford Myers' "The Art of Software Testing", we only end up in discussing different points of view, different beliefs. And the content is of second-order and therefore only discernable within the process it emerges in.

The logic of the Laws of Form makes explicit that no statement is to be taken existentially, which is something a tester should be familiar with.

REFERENCES

1. http://www.asc-cybernetics.org/foundations/definitions.htm
2. N. Wiener. Cybernetics or Control and Communication in the Animal and the Machine. MIT Press, Cambridge, 1948.
3. H. von Foerster. The Cybernetics of Cybernetics. BCL Publication, Urbana, 1974.
4. G. Spencer-Brown. Laws of Form. Allen and Unwin, London, 1969.
5. H. von Foerster. Ethics and second-order cybernetics. Understanding understanding: Essays on cybernetics and cognition, 2003.
6. R. Glanville. Why design research? Design: Science: Method, 1981.
7. N. Luhmann. Social Systems. Stanford University Press, Stanford, 1995.
8. D. Baecker. The writing of accounting. Stanford Literature Review, 1992.
9. G. Bateson. Steps to an Ecology of Mind. Chandler, New York, 1972.
10. G. Pask. The architectural relevance of cybernetics. Architectural Design, 7(6):494-496, 1969
11. S. Beer. Designing freedom. House Of Anansi, 1993.
12. W. Ashby. Introduction to Cybernetics. Chapman and Hal, London, 1956.
13. H. von Foerster and B. Pörksen. Wahrheit ist die Erfindung eines Lügners. Carl-Auer-Systeme, Verl. und Verl.-Buchh., 1998.
14. L. Wittgenstein. Tractatus Logico-Philosophicus. Routledge and Kegan Paul, London, second edition, 1971.
15. D. Baecker. Nie wieder Vernunft. Carl Auer Verlag, 2008.
16. G. Pask. Conversation, cognition and learning. Elsevier, New York, 1976.
17. M. Bolton. Testing vs. Checking, August 2009. URL http://www.developsense.com/blog/2009/08/testing-vs-checking/. [Accessed: 10-Apr-2012].
18. M. Bolton. "Merely" Checking or "Merely" Testing, November 2009. URL http://www.developsense.com/blog/2009/11/merely-checking-or-merely-testing/. [Accessed: 10-Apr-2012].
19. A. Turing. Computing machinery and intelligence. Mind, LIX(236), 1950.
20. R. Glanville. Chasing the blame. Research on progress - Advances in interdisciplinary studies on systems research and cybernetics, 11, 1995.
21. R. Glanville. The Question of Cybernetics. Cybernetics, an International Journal, 18, 1987.
22. J. Bach. Exploratory testing explained. 2003. URL http://www.satisfice.com/articles/et-article.pdf.
23. R. Glanville. The Value of being Unmanageable: Variety and Creativity in CyberSpace. 2000.
24. G. M. Weinberg. Quality Software Management, Volume 1: Systems Thinking. Dorset House Publishing Co., Inc., 1992.
25. M. Bolton. Done, The Relative Rule, and The Unsettling Rule, September 2010. URL http://www.developsense.com/blog/2010/09/done-the-relative-rule-and-the-unsettling-rule/. [Accessed: 10-Apr-2012].
26. F. J. Varela. Laying down a path in walking: A biologist's look at a new biology and its ethics. Human survival and consciousness evolution, pages 204-217, 1988.
27. N. Brunsson. The Organization of Hypocrisy: Talk, decisions and actions in organizations. John Wiley & Sons, 1994.

For example, the models of financial institutions and banks (before the financial crisis) were based on the Gaussian distribution. And just now we realise that a normal distribution is not necessarily the normal case. Second-order cybernetics and the Laws of Form (the logic of second-order cybernetics) help us to understand that the normal distribution should be viewed as a special case of several possible distribution patterns, in which extreme accumulations are possible as well.

Will a reflection session end up in conflict? It might, but that depends on the hiring process[viii] and on the organizational structure. Performing reflection sessions is easier in an organization that is based on conflict, not on agreement because "conflict as a principle of recruitment and organization [...] lead to reflection"[27]. Keeping conflict alive is a key element, but it is hard to do so because people who work together tend (over time) to think alike[ix]. But only if there are different views, thoughts and/or opinions within a team, a reflection session will be of value, because only then there is an exchange of ideas and the possibility to come to value new ways, new approaches.

Let me finish the article with a quote of Glanville[vi] that is, in my opinion, closely related to testing: "We may recognize a computer working as a medium… When we find it producing bizarre results and/or when we use it (meaning, essentially, its software) in an "incorrect" yet productive manner. Thus, we may look for distortions. But distortions, the unexpected, the unexplained, the unanticipated, the random are all more-or-less interchangeable terms that, in indicating surprise may also indicate novelty, the new." □

FOOTNOTES

i. "... it removes the claim of naive and absolute objectivity that we have so (recently damaging) built into our culture, our thinking and language, so that we deny experience and imagine (!) a universe that exists entirely of our imagining" [19]
ii. If I would have been consequent, the title of the article really should be: What I as a tester can learn from cybernetics (of cybernetics)
iii. Like the air conditioning on several German ICE trains, which lead to inside temperatures to up to 50 degrees Celsius
iv. Like Cem Kaner has done it with the BBST, which is now continued by the AST
v. http://twitter.com/testertested/status/130638007677108224
vi. Luhmann[7] insists that we cannot communicate, we can only take part in an communication.
vii. As far as cybernetics can be considered 'modern'
viii. See http://thesocialtester.co.uk/gravitate-to-people-like-you or Brunsson [27]
ix. See Argyris and Schön 1978, Jönsson and Lundin 1977

Check out our latest events and resources on www.ministryoftesting.com

CARTOON CORNER BY ANDY GLOVER A.K.A *THE CARTOON TESTER - http://cartoontester.blogspot.com/*

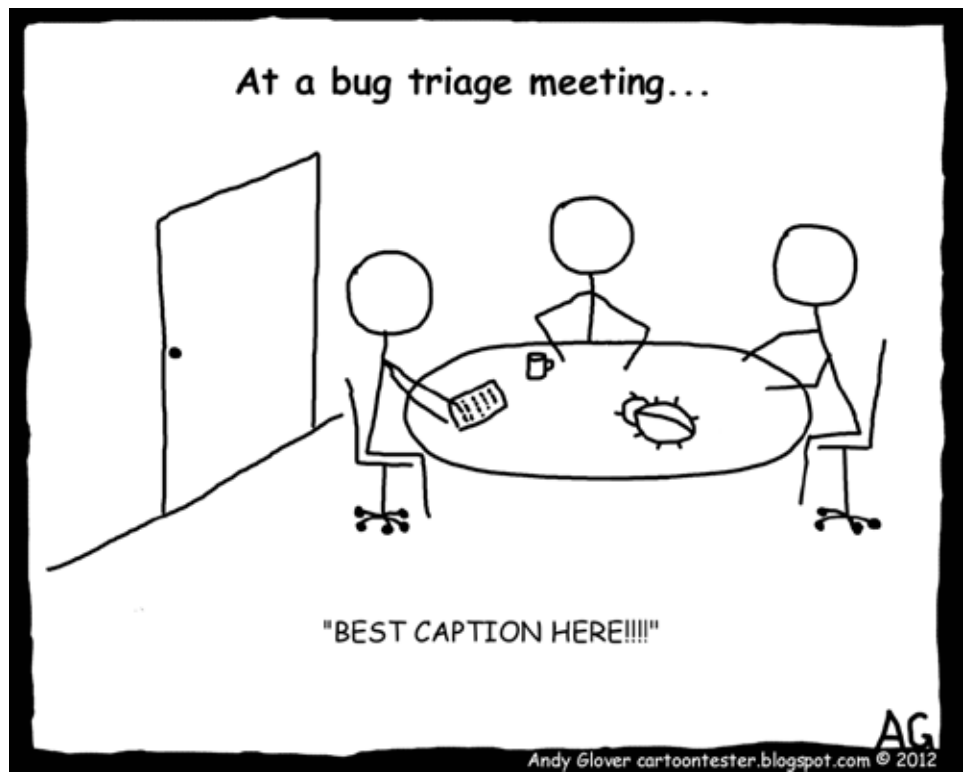
the cartoon corner

CAPTION COMPETITION!

WIN!

Send in your caption ideas for the cartoon below by the 31st September for a chance to win a free book! Submit your entry on The Testing Planet site (address below) and our resident cartoonist Andy Glover will judge your entries ASAP after the closing date. Good Luck!

http://www.thetestingplanet.com/2012/07/bug-triage-caption-competition/



Your one page guide to Session Based Test Management - http://bit.ly/sbtmmindmap

elevate™

The direct recruitment platform connecting employers and contractors

Cut out the middle man today!

Employers

Reduce your costs - pay **just 4% mark-up** on day rate, 6% for our payroll end-to-end solution.

Reduce AWR overheads by directly engaging with the talent you need.

Track workflow and manage the hiring process for **improved efficiency**.

Enable oversight, unified views and control of hiring, renewals and activity over time.

Contractors

Find client work without reliance on recruitment agencies and third parties.

Apply directly for roles and receive direct approaches from employers with suitable requirements.

Manage your applications through our transparent interview and offer process.

Use elevate for free and keep 100% of your rate.

GET STARTED *to pay* **JUST 4% MARK-UP!**

SIGN UP FREE *to see* **YOUR MATCHING JOBS**

Great roles available now from leading employers

wazoku ocado GALA CORAL GROUP TATA NMQA CAKESOLUTIONS

www.elevatedirect.com

Lean startup: An interview with Ben Wirtz

INTERVIEWEE PROFILE - BEN WIRTZ

Benjamin runs Handy Elephant, a Citrix Startup Accelerator funded company based in Cambridge, UK. Before getting investment, he founded and bootstrapped Handy Elephant single-handedly and was a freelance Android developer. His background is in information systems, with a passion for social computing and relationship management. Before becoming an entrepreneur, he studied in Germany and England, worked in IT consulting and had a job as assistant sail instructor in the Whitsundays/Australia.

WHAT WERE YOU DOING BEFORE YOU STARTED YOUR BUSINESS VENTURE?

I started Handy Elephant out of my masters in E-Business & Innovation.

WHERE DID YOU COME UP WITH THE IDEA AND HOW HAS THE BUSINESS EVOLVED?

The original business idea was very different. I wanted a better tool to organise notes on contacts and meetings, which would also remind me to follow up with people. The past 1½ years have been about testing our assumptions on the problem and potential solutions against the market. We have seen many different products related to contact and relationship management by (often well-funded) competitors along the way, which gave it one big shot and ultimately failed. Our prototypes became better with each iteration and as a result we are now in a position to release a product that we have been offered money for.

WHAT'S THE BIGGEST CHALLENGE OF RUNNING A SOFTWARE COMPANY?

Work-life balance. At first, it's easy to think that you will succeed if you work hard enough, and you can make a lot of progress. But after what might seem like a never-ending slog you can start to lose focus. It's easy to end up running in the wrong direction without even realising it.

I think it's important to always have a fresh mind by learning new things, meeting new people and being interested and engaged in a diverse range of topics and activities. If you do this, the ups and downs of running a company may not hit you as hard. It's easier said than done though and I still struggle with it after all this time. I've heard it said that having one fixed day per week in which you don't do anything for your business at all is the least you should do.

CAN YOU TELL US ABOUT HOW YOU USED LEAN PRINCIPLES TO BUILD YOUR BUSINESS?

I've been a fan of the Lean methodology since the first Start-ups Lessons Learned[1] conference in 2010, but it's much harder to implement than it sounds. In my experience when people say they only use part of it, they are probably not Lean. Learning definitely has to be at the centre of everything you do. The main problems I have found with the Lean start-up methodology is actually being able to identify my own assumptions; things that seem like common sense. It's also necessary to test the riskiest assumptions first and doing this means having to define a good experiment, something that also is quite difficult!

I see these as being core principles of the Lean approach. If you don't get them right then everything else that people have in mind when thinking about their tech' start-up (using open source software, automated tests and continuous integration, having lots of "actionable analytics"[2], releasing buggy software, not raising investment) doesn't even matter – those are just tools. In the early stages, actually building software should usually be the last resort; there are usually quicker ways to verify your ideas, e.g. interviewing customers or non-functional clickable prototypes.

During the lifespan of my business so far, we have made all the mistakes of a startup that thought it was a lean-startup. I can confidently say we're not experts but we certainly still have the aspiration to be lean and we will keep learning how to achieve this.

HOW DO YOU THINK A TESTER CAN ADD VALUE TO A START-UP?

I'm convinced (after speaking to testers at TestBash[3]) it is the tester mind-set. These days most companies will hopefully write automated tests for their code and that might be done by either developers or dedicated testers. It's been said though that developers often think more about the actual execution ("How can I make this happen?") whereas testers will think about validation ("What is the expectation and have we delivered it?"). These questions and the skills to answer them are useful not just in software testing, but also in testing other hypotheses, business models and marketing strategy for example.

Eric Ries has defined the build-measure-learn cycle[4] as the core element of a Lean Startup methodology. While developers could build stuff all day long and business people would love to measure just anything, with a testing mindset onboard, there might be more focus on (a) what assumptions are we trying to verify, and (b) what is the quickest way to verify them.

WHAT DOES YOUR TEAM LOOK LIKE? AT WHAT STAGE DO YOU THINK YOU MIGHT HIRE A TESTER TO HELP OUT?

We are 3 hackers:
1 software development hacker
1 UX & design hacker
1 growth hacker

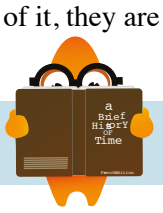
At this stage, it seems everyone needs to have a bit of a testing mentality, so hiring a dedicated test-hacker will come at some stage - but we wouldn't be lean if we had an exact plan for when exactly that is!

IF A CUSTOMER NEVER SEES A BUG, DOES IT EXIST?

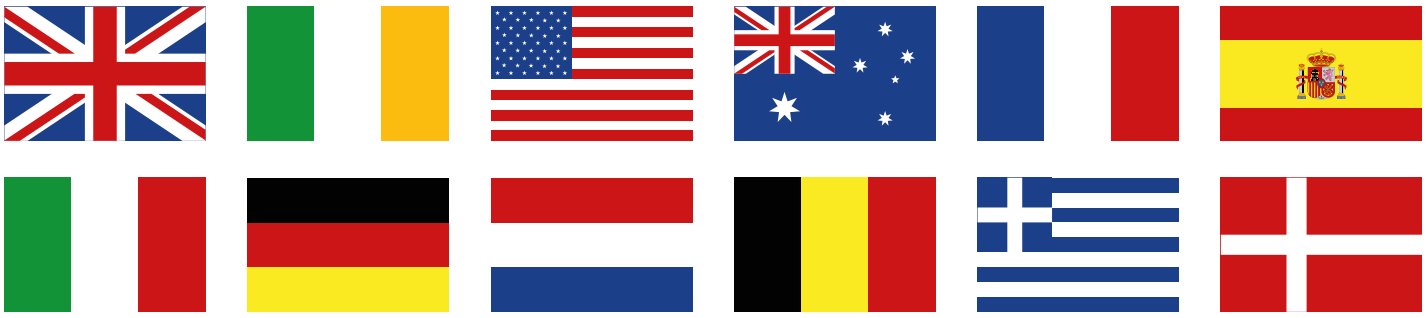
It's not a bug, it's a test! □

REFERENCES

1. http://www.sllconf.com/
2. http://www.fourhourworkweek.com/blog/2009/05/19/vanity-metrics-vs-actionable-metrics/
3. http://www.ministryoftesting.com/training-events/testbash/
4. http://the leanstartup.com/principles



What should you be thinking about when testing for accessibility? Check out this MindMap and list - http://bit.ly/accessibilitymm



Quality isn't something, it provides something

By James Christie

A few weeks ago two colleagues, who were having difficulty working together, asked me to act as peacekeeper in a tricky looking meeting in which they were going to try and sort out their working relationship. I'll call them Tony and Paul. For various reasons they were sparking off each and creating antagonism that was damaging the whole team.

An hour's discussion seemed to go reasonably well; Tony talking loudly and passionately, while Paul spoke calmly and softly. Just as I thought we'd reached an accommodation that would allow us all to work together Tony blurted out, "you are cold and calculating, Paul, that's the problem".

Paul reacted as if he'd been slapped in the face, made his excuses and left the meeting. I then spent another 20 minutes talking Tony through what had happened, before separately speaking to Paul about how we should respond.

I told Tony that if he'd wanted to make the point I'd inferred from his comments, and from the whole meeting, then he should have said. "your behaviour and attitude towards me throughout this meeting, and when we work together, strike me as cold and calculating, and that makes me very uncomfortable".

"But I meant that!", Tony replied. Sadly, he hadn't said that. Paul had heard the actual words and reacted to them, rather than applying the more dispassionate analysis I had used as an observer. Paul meanwhile found Tony's exuberant volatility disconcerting, and responded to him in a very studied and measured style that unsettled Tony.

Tony committed two sins. Firstly, he didn't acknowledge the two way nature of the problem. It should have been about how he reacted to Paul, rather than trying to dump all the responsibility onto Paul.

AUTHOR PROFILE - JAMES CHRISTIE

James is a software-testing consultant based in Perth, Scotland. His website is http://clarotesting.com/ and blog is http://clarotesting.wordpress.com/. He can also be followed on Twitter, @james_christie. With 27 years commercial IT experience, in addition to testing he has worked in information security management, project management, IT audit, systems analysis and programming. This experience has been largely in financial services, but has covered a wide range of clients, throughout the UK, and also in Finland.

Secondly, he said that Paul is cold and calculating, rather than acting in a way Tony found cold, and calculating at a certain time, in certain circumstances.

I think we'd all see a huge difference between being "something", and behaving in a "something" way at a certain time, in a certain situation. The verb "to be" gives us this problem. It can mean, and suggest, many different things and can create fog where we need clarity.

Some languages, such as Spanish, maintain a useful distinction between different forms of "to be" depending on whether one is talking about something's identity or just a temporary attribute or state.

The way we think obviously shapes the language we speak, but increasingly scientists are becoming aware of how the language we use shapes the way that we think.[1]

The problem we have with "to be" has great relevance to testers. I don't just mean treating people properly, however much importance we rightly attach to working successfully with others. More than that, if we shy away from "to be" then it helps us think more carefully and constructively as testers.

This topic has stretched bigger brains than mine, in the fields of philosophy, psychology and linguistics. Just google "general semantics" if you want to give your brain a brisk workout. You might find it tough stuff, but I don't think you have to master the underlying concept to benefit from its lessons.

Don't think of it as intellectual navel gazing. All this deep thought has produced some fascinating results, in particular something called E-prime, a form of English that totally dispenses with "to be" in all its forms; no "I am", "it is", or "you are". Users of E-prime don't simply replace the verb with an alternative. That doesn't work. It forces you to think and articulate more clearly what you want to say.[2]

"The banana is yellow" becomes "the banana looks yellow", which starts to change the meaning. "Banana" and "yellow" are not synonyms. The banana's yellowness becomes apparent only because I am looking at it, and once we introduce the observer we can acknowledge that the banana appears yellow to us now. Tomorrow the banana might appear brown to me as it ripens. Last week it would have looked green.

You probably wouldn't disagree with any of that, but you might regard it as a bit abstract and pointless. However, shunning "to be" helps us to think more clearly about the products we test, and the information that we report. E-prime therefore has great practical benefits.

Continued on page 14

Is it ethical to hide (unrepeatable or unimportant) bugs - http://bit.ly/ethicalbugs

Continued from page 13

The classic definition of software quality came from Gerald Weinburg in his book "Quality Software Management: Systems Thinking". Quality is value to some person".

Weinburg's definition reflects some of the clarity of thought that E-prime requires, though he has watered it down somewhat to produce a snappy aphorism. The definition needs to go further, and "is" has to go!

Weinburg makes the crucial point that we must not regard quality as some intrinsic, absolute attribute. It arises from the value it provides to some person. Once you start thinking along those lines you naturally move on to realising that quality provides value to some person, at some moment in time, in a certain context.

Thinking and communicating in E-prime stops us making sweeping, absolute statements. We can't say "this feature is confusing". We have to use a more valuable construction such as "this feature confused me". But we're just starting. Once we drop the final, total condemnation of saying the feature is confusing, and admit our own involvement, it becomes more natural to think about and explain the reasons. "This feature confused me … when I did … because of..."

Making the observer, the time and the context explicit help us by limiting or exposing hidden assumptions. We might or might not find these assumptions valid, but we need to test them, and we need to know about them so we understand what we are really learning as we test the product.

E-prime fits neatly with the scientific method and with the provisional and experimental nature of good testing. Results aren't true or false. The evidence we gather matches our hypothesis, and therefore gives us greater confidence in our knowledge of the product, or it fails to match up and makes us reconsider what we thought we knew.[3]

Scientific method cannot be accommodated in traditional script-driven testing, which reflects a linear, binary, illusory worldview, pretending to be absolute. It tries to deal in right and wrong, pass and fail, true and false. Such an approach fits in neatly with traditional development techniques, which fetishize the rigours of project management, rather than the rigours of the scientific method.

This takes us back to general semantics, which coined the well-known maxim that the map is not the territory. Reality and our attempts to model and describe it differ fundamentally from each other. We must not confuse them. Traditional techniques fail largely because they confuse the map with the territory.[4]

In attempting to navigate their way through a complex landscape, exponents of traditional techniques seek the comfort of a map that turns messy, confusing reality into something they can understand and that offers the illusion of being manageable. However, they are managing the process, not the underlying real work. The plan is not the work. The requirements specification is not the requirements. The map is not the territory.

Adopting E-prime in our thinking and communication will probably just make us look like the pedantic awkward squad on a traditional project. But on agile or lean developments E-prime comes into its own. Testers must contribute constructively, constantly, and above all, early. E-prime helps us in all of this. It makes us clarify our thoughts and helps us understand that we gain knowledge provisionally, incrementally and never with absolute certainty.

I was not consciously deploying E-prime during and after the fractious meeting I described earlier. But I had absorbed the precepts sufficiently to instinctively realise that I had two problems; Tony's response to Paul's behaviour, and Paul's response to Tony's outburst. I really didn't see it as a matter of "uh oh – Tony is stupid".

E-prime purists will look askance at my failure to eliminate all forms of "to be" in this article. I checked my writing to ensure that I've written what I meant to, and said only what I can justify. Question your use of the verb, and weed out those hidden assumptions and sweeping, absolute statements that close down thought, rather than opening it up. Don't think you have to be obsessive about it. As far as I am concerned, that would be silly! □

REFERENCES

1. "How Language Shapes Thought", Lera Boroditsky, Scientific American (February 2011) http://psych.stanford.edu/~lera/papers/sci-am-2011.pdf
2. "Speaking in E-prime: An Experimental Method for Integrating General Semantics into Daily Life", E W Kellogg III, Et cetera Vol 44, No 2, 1987. http://www.generalsemantics.org/wp-content/uploads/2011/05/articles/etc/44-2-kellogg.pdf
3. "Working with E-prime – Some Practical Notes", E W Kellogg III & D David Bourland Jr, Et cetera Vol 47, No 4, 1990. http://www.asiteaboutnothing.net/pdf_workingwitheprime.pdf
4. "The Map is not the Territory", Less Wrong wiki community blog. http://wiki.lesswrong.com/wiki/The_map_is_not_the_territory

Lean startup: An interview with Alan Downie

THERE IS THIS LIGHTBULB MOMENT, WHERE YOU HAVE A BUSINESS IDEA AND YOU WANT TO MAKE IT HAPPEN. HOW DO YOU GET IT TO BECOME REALITY?

A lot of people in this industry have lightbulb moments daily. The trick is to know which ones to ignore, and which ones to get excited about. The first thing I actually recommend is to do "nothing"! Often that great idea doesn't seem so great a few days later. You need to sit and think on it a while. Let it fester and build and annoy you when you shower, when you drive to work and when you sleep. Only when you can't get it out of your head after a week or two should you start thinking about actually turning it into a business.

Then the first thing to do is to validate whether anyone else is as excited about your idea as you are... And more importantly, would they pay for it? Everyone is crazy at the moment on going off and building an MVP (Minimum Viable Product), but they tend to forget the Viable part. Before you build anything you need to confirm to yourself that there is actually a business behind your idea.

Once you have some keen customers lined up you can start actually building the product, sourcing their feedback early and often. The goal should be to get to a paid product as soon as possible, and validate that these customers' enthusiasm will actually translate to a payment.

Continued on page 15

Looking for a new job? Here are some tips on how to get hired - http://bit.ly/gethiredmindmap

Continued from page 14

A lot of people will try and pay with kind words and smiles, but it won't pay your mortgage. If you really are onto something great, people will literally throw money at you to get involved.

DO YOU USE BUGHERD INTERNALLY?

I have to be honest here; the answer is not always! The problem with BugHerd is that it isn't capable of tagging itself. Our tool is designed so that a user can't accidentally tag our app instead of their own. The down side of that means we can't tag errors in our app either! We do however use BugHerd as our bug tracker (even without the bug tagging). We run our sprints using Pivotal Tracker (which integrates with our bug list in BugHerd), we use our own integration with GitHub and also with Zendesk for support queries. We're our own best customers!

WHAT HAVE BEEN THE BIGGEST AND/OR MOST CHALLENGING BUGS YOU'VE HAD IN YOUR APP?

The hardest part of building an app like ours is that for the most part it is running on someone else's website. It means not only do we have to contend with a bunch of browser issues, cross-domain communication issues, but we also need to make sure we don't interfere with our customers' websites. This has meant putting a lot of time into sandboxing our app and doing a lot of really crazy performance testing to minimise the impact on the load times of both our app and their website.

DO YOU HAVE 'PROFESSIONAL' TESTERS IN YOUR TEAM? WHY/WHY NOT?

[A] Not yet unfortunately. We're a small team of 2 developers and 1 designer. Our next hire is likely to be a support/tester role though. As I mentioned, our app runs across so many different environments it's critical for us to get this right. We do however have a pretty cool functional testing suite for our app.

DO YOU RELY ON YOUR CUSTOMERS TO REPORT BUGS? IF SO, HOW DOES THIS AFFECT YOUR RELATIONSHIP WITH THEM - IS IT A GOOD OR BAD THING, GENERALLY?

We do a lot of testing internally before we do releases, and we almost always release new features to a small group of beta customers first to minimise the chances of bugs getting live, but of course it does happen occasionally. I actually love it when customers report bugs. Of course I wish we didn't have bugs in the first place, but if someone has taken the time to report an issue it means they actually care. As a startup, getting people to care about your product is the biggest challenge you'll face. If someone takes the time to report an issue to us, we make sure to show them how grateful we are for their efforts.

WHAT'S THE HARDEST THING ABOUT RUNNING A TECH BUSINESS?

The hardest thing about running a startup is finding that elusive product/market fit. You know you have a great product, and you know people want it; you just need to find the market that gets the most value out of your idea (and thus pays you the most money!), and then find a way to get it in front of them. Anyone can build a product; it's another thing to build a product people will actually pay for! □

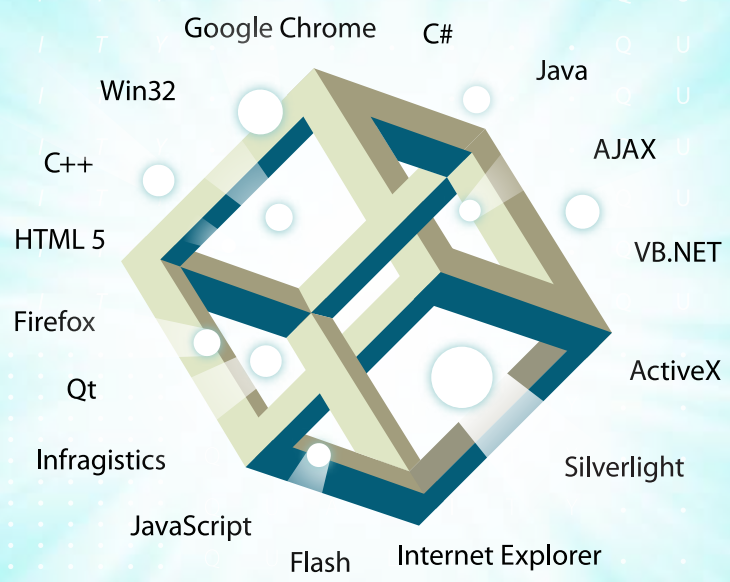
INTERVIEWEE PROFILE - ALAN DOWNIE

Alan Downie is the Co-founder and CEO of BugHerd, the point and click bug tracker for the web. Alan has been working in web for over 15 years. He headed up development of the award winning Intranet DASHBOARD before teaming up with Matt Milosavljevic to produce the popular UsabilityHub, Fivesecondtest and BugHerd.

One Tool to Test Them All

- Web, Windows, and Java testing
- Functional, stress, and load testing
- Object-based recording and intelligent scripting

Get your free QA Wizard Pro trial now!

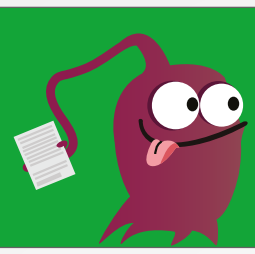


www.seapine.com/qawizard.html

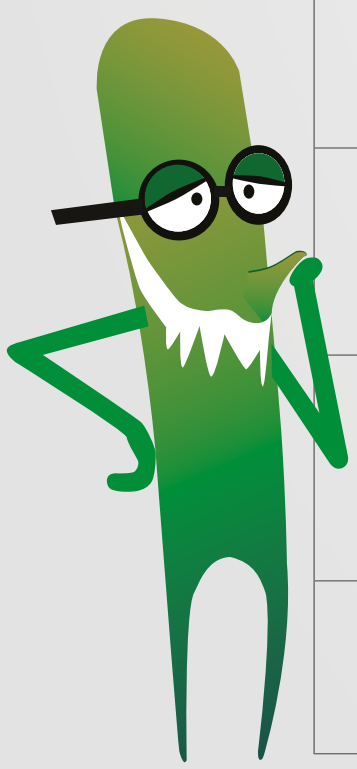
Seapine SOFTWARE

HOW LEAN IS Y

Lean software development is gaining support but how does that affect your test
but we should all be following the lean principles of 'Seeing the whole picture' and
this chart to help you decide if you're using the lea

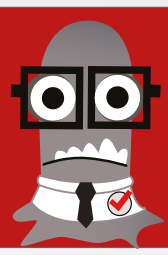
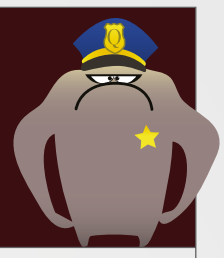


FEATURES OF A TESTER	COWBOY	LEAN	AGILE
DOCUMENTATION	What documentation?	Automated tests written before and during development which later serve as documentation (ATDD)	Automated tes development Manual testing is light-weight, easy c such as mind-ma
TOOLS	What tools?	Lightweight tools that can be quickly set up and learnt	Bug manag
ROLE	I'm only a tester in my spare time	Likely to involve tasks outside of traditional testing: user support, coding, marketing etc	Dedicated tester team i.e. tester
LEARNING	Hard Knocks!	Peer Knowledge Swap Hard Knocks! Internet / Blogs / Communities Books	Peer Know Internet / Blogs Bo
TEST PLANNING	We don't plan testing	Just in time	Scheduled b
RELEASE SCHEDULE	Code and push. Repeat to fix everything that breaks Releases are frequent and form part of the ongoing development and release cycle	Frequent. Releases probably not scheduled but instead shipping as soon as they are 'ready' Releases are frequent and form part of the ongoing development and release cycle	Frequent release Releases are fr part of the ongo and rele
BUG PRIORITISATION	Unlikely to happen. Bugs picked up and fixed as developers wish	Frequently re-prioritised against features	Severity and pri room to re-pr release sched
BUG TRACKING	Bugs don't need tracking - just get 'em fixed!	Bugs raised by pretty much everyone Physical bug reports (index cards, post-it notes)	Bugs raised by p well as develop Bugs recor manage
GOAL	Get this code live	Quick releases to get feedback from users. Testing is complete when the Minimal Viable Product (MVP) is usable	Maintaining as bugs as possib environment. R favoured above n



OUR TESTING?

ting? Different organisations and projects require different approaches to testing
d 'Building Integrity in'. Could you 'eliminate waste' and 'empower the team'? Use
nest possible approach to testing for your project.

		V-MODEL 	TOTALLY ENTERPRISE 
	ts written before begins (ATDD) documented using changeable test plans aps or Google docs	Integration test plan and System test plan written using design documents. Unit and Integration tests created but less likely to form business facing documentation	Do it by the book. Make sure you have Test Policies, strategies and test plans written and signed off before testing begins. Test entry and exit criteria should be documented
	gement tool	Test management tool Bug management tool	Test management tool Bug management tool Time management tool
	r within mixed role on a scrum team	Dedicated tester within a test team. System testing and Integration testing are clearly defined phases and may involve different teams of testers	Multiple test teams are usually involved to cover integration, system, security, performance and acceptance testing. Off-shore is probably the norm
	wledge Swap s / Communities ooks	Books Formal Training Courses	Formal Training Courses
	ut fast paced	Formal. Clearly defined test analysis and execution phases	Very formal. Dedicated team members to plan and estimate testing phases
	t, planned schedule equent and form oing development ease cycle	Infrequent. Well defined with clear development and test phases Release is likely to indicate completion of the project	Rarely. Releases are a very big deal Release is likely to indicate completion of the project
	iority defined but ioritise to meet dules if needed	Clearly defined priority and severity ratings. Classifications are usually part of a company wide standard. Testing phases will be extended if pre-agreed levels of bugs are exceeded	Bugs reported and classified as defined in industry defined standards
	roduct owners as bers and testers rded in a bug ment tool	Bug reports coming mostly from the testers Recorded in a test management system and likely to be linked to test plans	All bugs are raised by testers Recorded in a test management tool and linked to test plans, requirements, technical specs etc
	few production le in an iterative egression testing new feature testing	Aiming for no bugs in production	Aiming for no bugs in production plus a usable, secure, functionally valid and performant system

ATDD (Acceptance Test Driven Development):

A collaborative activity where the whole team works to produce Acceptance Criteria with examples before the development begins. The goal is to create a shared understanding of the product or feature.

MVP (Minimal Viable Product):

Frequently used in Start-ups to define the features needed for launch and nothing more. Popularised by Eric Rees.

Lean development principles:

Eliminate Waste, Amplify Learning, Decide as late as possible, Delivery as early as possible, Empower the team, Build Integrity in, See the whole picture.

By Rosie Sherry & Amy Phillips

The 5 W's of Lean Requirement Analysis

By Dan Ashby

I have come to realise, through my experience and from discussion with other testers, that a vast amount of project cycles (almost 65% out of all the projects that I have worked on), have a lack of consideration for the requirement-gathering phase of the project. I understand that some customers might not supply a requirements document, and that an exploratory testing method could be used to obtain an understanding of what the software is and should do, but for those projects where requirement documents are supplied, there does not seem to be enough emphasis on the need to test them. I've seen many instances where requirements have appeared very vaguely written. In the book "An Information Systems Manifesto", James Martin publishes his Distribution of Defects, showing that 56% of defects found have a root cause of poorly defined requirements. The meta-information available from this statistic certainly rings true in my own experience.

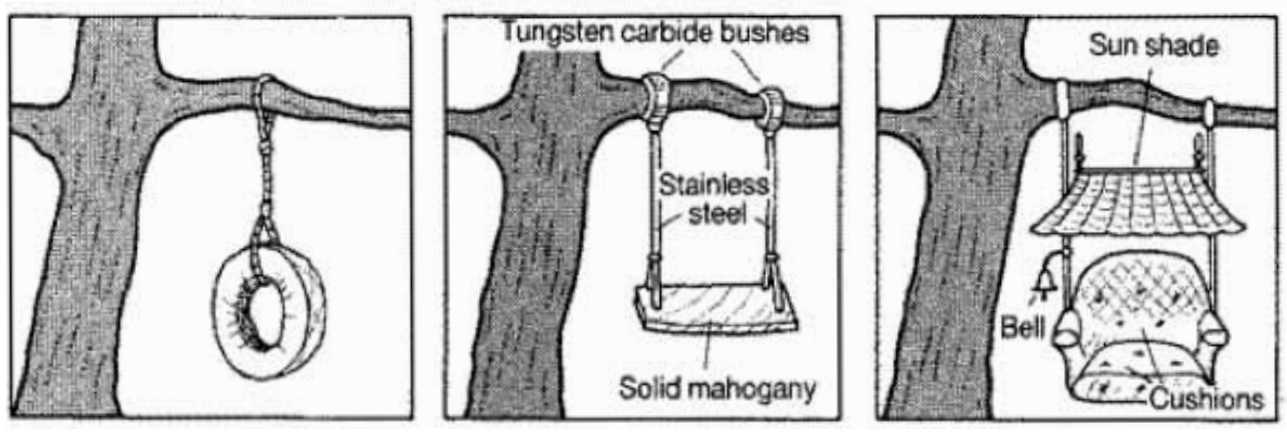
Why should requirement documents be tested?

I'm sure we have all seen our fair share of customers that simply supply a minimalist specification that is full of one-liner requirements, that the developers start working on while the testers start writing test cases. There have been many situations where the developed product hasn't really met what the customer actually wanted. Many of these have been captured via stories on the web about failed projects due to the developed system not meeting the customer's needs. One such story can be found in Lorin J. May's article "Major Causes of Software Project Failures", where a multi million pound military project, dubbed the "Titanic of military projects", ended in total failure when users refused to use the system due to it lacking essential features for them to be able to do their jobs.

There are many articles and blog posts online that say: "Most bugs in software are due to inaccurate functional requirements". In fact, the meaning behind James Martin's statistics from his research backs this up, but I'm sure most of us have experienced these situations.

Ultimately, ambiguity in the requirements cost us all time and money. Spending time to develop something that is wrong from the start is very frustrating.

Too often, Testers are left out of the requirement-gathering phase, but in reality this is where the testing process should begin! In every



What the customer specified "We want a tree swing"

What the developers produced "Here is your amazing tree swing"

What the customer actually wanted "No! This is what we wanted!"

Your software development process?[1]

development project regardless of the development methodology in place, the initial requirement-gathering phase is very important - this is where the product is defined. So it's essential that each requirement is tested. Questions need to be asked of each requirement to dispel any uncertainties from what the customer is trying to say and also to fill any gaps in the requirements, so that we can gain some confidence in being able to develop the product in line with what the customer actually wants. Discovering additional requirements that the customer has missed can be such a satisfying feeling too.

Why is Requirement Analysis LEAN?

LEAN is about being smarter and more cost effective in the project cycle to reduce waste and optimise the processes within the said project. This should in turn increase the quality of the product and the customer's satisfaction in the product. In a time where more and more companies are trying to be leaner, the principles of "LEAN" can definitely be applied to the testing phases of the project in many ways! One way is to reduce testing cost and effort by testing as early as possible in the project cycle, which means testing the requirements...

Performing requirements analysis should clarify and dispel any assumptions in the requirements and reduce the amount of defects found in the product. This will enable the developers to build software that aligns more closely to what the customer actually wants. This also means that there will be less time, money and effort spent on the re-development and testing of the software later in the project cycle, due to the change requests that the customer would otherwise need to submit because of the software being based on incorrect requirements.

So how do we perform Requirements Analysis?

Testing is all about asking questions. Whether it is software, hardware or documents being tested, testing is about asking probing questions of it. Turning assumptions into facts. Trying to gain an understanding of what it does and what it's supposed to do, to then be able to determine whether or not it meets the customer's needs.

When the customer is collating requirements, the main aim of testing is to make sure that each requirement is understood (not only by the developers and testers, but by the customer too - they might have missed something very important from the specification that they never previously considered!) We also need to ensure that each requirement is specified in such a way that ensures we are confident the customer wants what has in fact been written. As a basis, the tester should try to find the answers for each of the following "5 W" questions for each requirement: Who, What, Where, When and Why? Additionally, when asking these, we need to think about them in both a positive AND negative context!

An example - Say the product was a recruitment web application and one of the requirements states: "An applicant needs to be able to submit an application form for a job post". We need to ask appropriate questions based on this requirement in order to gather more information on

Continued on page 19

We have a job board for testing jobs - http://jobs.softwaretestingclub.com

Continued from page 18

what the customer actually means by the statement. We want to dispel all possible assumptions from the requirement:

WHO:
- Who should be able to submit/edit the application (e.g. registered users only)?
- Who will receive the application form?
- Who should be able to view the submitted form (e.g. in a read only format)?
- Who should receive any email notifications to say that an application has been submitted?
- Who should NOT have access to the application form screen to be able to submit an application (e.g. a non registered user)?
- Who should NOT be able to submit this specific application form (e.g. someone who has already applied?)

WHAT:
- What fields should be available on the application form?
- What relationships should fields have with each other (e.g. should this field auto-populate that other field)?
- What about default values in the fields?
- What validation should be in place (e.g. any mandatory fields or any field limitations)?
- What validation messages should be displayed?
- What buttons should be available for the user?
- What should the notification say?
- What should happen when the application form is submitted?
- What should NOT happen when the application form is submitted?

WHERE:
- Where should the new application form be in the system (e.g. should it load immediately after the user logs in)?
- Which screens should allow access to the form (e.g. should a link to the application form be available from any other screens)?
- Where should the link be located on these screens?

WHEN:
- When in the workflow process of the system should the application form be available to the applicant?
- When in the workflow process should the submitted application form be available to the receiver of the application?
- Do certain other actions need to be performed before the application form becomes available?

WHY:
- Why is the new function being implemented this way (e.g. is it consistent with other functions for the product)?
- Is this the best way to implement this new function?
- Are there any improvements that can be made to the way that this function is suggested to work (e.g. less button clicks, etc.)?

Asking questions like these should help the customer to think about their expectations of each requirement in a way that they may not have done previously. They will also serve to dispel assumptions made within the requirements and help to identify specification gaps.

Final Thoughts

Requirements should ideally be clear and concise with minimal uncertainty or assumptions, and should be complete without any contradictions.

Testing the requirements does mean that more time must be spent during the requirement gathering process. However, requirements analysis is in fact very lean as it will save money, time, and effort later in the project cycle. If requirements analysis is not performed, then the cost of correction escalates for any changes that the customer or the project team have to make due to assumptions that have been made on ambiguous requirements.

Next time you find yourself analysing requirements, give the 5 W's a go. I would be very pleased to hear from you regarding the difference that it made. □

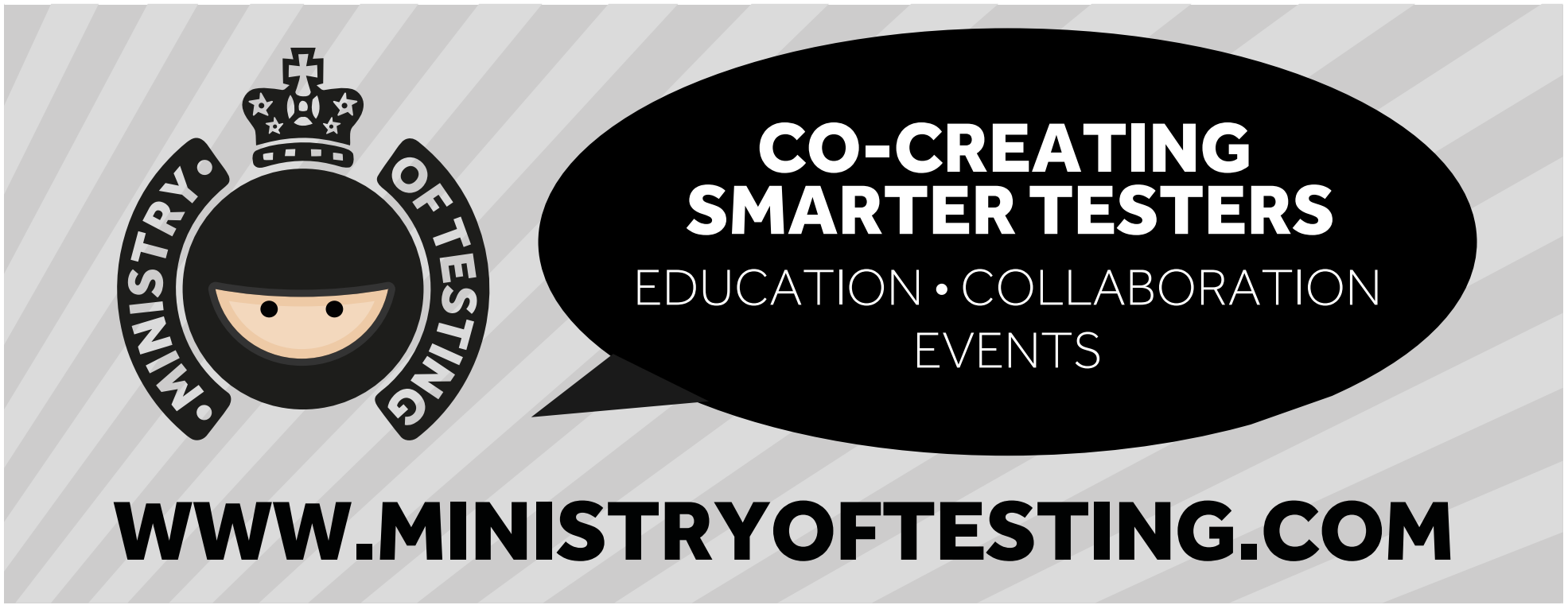
AUTHOR PROFILE - DAN ASHBY

Dan is a Software Test Analyst based in the London area, with over 7 years of experience in the testing industry. He has a passion for Exploratory Testing and is currently focused on testing web based applications and web sites. He is also interested in Automated Testing and Web App Security Testing. Dan regularly uses social media sites to connect with other testers and he loves being involved in discussions relating to anything testing and also being part of the testing community.

LinkedIn profile: http://uk.linkedin.com/in/danielashby
Twitter profile: @DanAshby04

REFERENCES

1. Image from http://www.devsource360.com/freedownload/free-tutorials/otherdownload/free-download-huge-collection-of-programming-cartoons.html



Write something for The Testing Planet! http://www.thetestingplanet.com/submit-content/

The curious case of the context-driven conference: Case notes

By Duncan Nisbet

Left: Julian Harty on open source testing / Right: Ola and his committee

I was hired by a client to investigate some curious goings on over in Sweden. Rumour had it that an infamous mob going by the name of "The Gang of Five" was putting together a context-driven testing conference. They claimed that it was by Testers, for Testers. Well my client wasn't so sure and I was inclined to agree, so I high-tailed it over to Sweden to see if the rumours were true…

The joint they chose to host this conference was in a nice secluded spot, surrounded by water, away from prying eyes and still only an hour way from the airport. Camouflaged as a university campus with a big hall for the keynotes, smaller lecture theatres for the sessions, tutorials and en-suite bedrooms in small blocks resembling halls of residence; the conference could have gone completely unnoticed by the casual passer-by, but not by me. Following my keen nose for a story, I decided to investigate further.

I had timed my arrival so the conference was in full swing. I got me some hooch and slunk down in a chair to monitor some chinning from Scott Barber and other plugs.

This turned out to be a great start to my investigation. I'd elicited information from a reliable source that Scott would open proceedings with a keynote speech alongside Michael Bolton, Rob Sabourin and with Julian Harty bookending the conference. Obtaining privileged information from one of the keynote speakers early on would substantiate whether the conference was for real or just baloney.

Chow was spent with me paying close attention to a conversation between Rob and Scott about the current situation of testing. It appears that some in the industry believe that testing is dead. That sounds like another case entirely…

So far, this conference was living up to the rumours – people who appeared to be Testers surrounded me. I'd need to bump gums with more of them to see if they were actually just ringers looking for a sucker.

I caught glimpses of other Testers I recognised from reconnaissance prior to attending the conference, including the big cheeses of the mob. I approached a couple of them to bump guns and to test my cover. It appears my shorts were lacking as a viable disguise, but they promised not to bust my chops.

The content of the conference had a wide range of topics for all kinds of Testers – which should I attend? My plan was to chew gum with as many Testers as possible to get to the bottom of case before the conference ended.

Each day was kicked off with a keynote for everyone, so no choice there. They all were nifty and provided many leads for my investigation. With everyone all in one room, it was a great opportunity for me to identify the ringers and grifters.

The first day was focussed on tutorials – getting stuck in with some testing. I decided I best spread myself thinly and commit to 2 half day sessions. The first was "Critical Thinking for Testers" with Michael Bolton. This guy had some great notions about thinking on your feet, but not necessarily going with your gut instinct. The seconds was "Now What's Your Plan" with Henrik Andersson and Leo Hepis. Changing their stories more often than their underwear, they would easily avoid being pinched!

The second day included 4 tracks of hour long sessions. The choice here was overwhelming – were they trying to hide something in plain sight? Each session would really help debunk any baloney, but which ones to attend? Options ranged from the more formal topics of testing in the financial industry and strategies for successful Systems Integration Testing through to the more obscure hypnotic methods of testing and the similarities between Testers and Art Critics. What an array of topics – how could they all be possibly related to testing?

I got a slant at what I felt were the more suspicious options: "Charter My Tests", "Testing Hypnotically", "Making the Case for Aesthetics" and "Coaching Testers". I was sure they had to be hiding something in there and I needed clues.

As well as a packed daytime itinerary, the mob had also put together a distracting evening schedule. This meant more interaction; more opportunity for people to tip their mitt and spill the beans. Options ranged from more sessions, puzzles, gatherings in the juice joint and guided art tours.

Naturally I was taking a keen interest in Alan Richardson (you might know him by his pseudonym Evil Tester) so I decided to attend the "Hypnosis Explained" session as well Michael Hunter's session on his "You are not done yet" source of test ideas. Aside from these sessions, I chose to mingle between the numerous rooms to eavesdrop on some of my marks and silently gather more intelligence for my investigation.

My fear that the sponsors might try and jeopardise my cover by putting me out on the roof turned out to be unfounded. They were actually just giving away free hooch accompanied by some great loot. Very curious indeed…

It's time to hang up my gumshoes and wrap up the first part of my investigation by saying this conference was no scam. After beating my gums with the mob I knew they were Testers – so they weren't trying to pull a fast one. The joint they chose was classy and I rated the speakers and content as hitting on all eight!

Apparently the conference is being run again next year. Was the conference this year just a ruse to get us thinking everything was on the level? There's only one way to find out… □

AUTHOR PROFILE - DUNCAN NISBET

Duncan Nisbet is an independent PI for hire with a keen desire to learn about all things software testing and development. No case too big, no case too small, when you need help, just call! His case notes can be found at http://www.duncannisbet.co.uk

Testing for Kids - Free Download - http://bit.ly/testingforkids

EuroSTAR Software Testing Conference

AMSTERDAM RAI
5 – 8 NOVEMBER 2012

Don't miss the Early Bird Discount book by 21 September

Test Lab
Real systems and real tools to test in a live & interactive environment!

Networking
Network & make friends with hundreds of industry peers. Mingle in the Expo and at social events!

Personalized Learning Experience
You pick what you want to attend out of an amazing 40 track sessions, 6 full day tutorials, 5 half day tutorials, 5 keynotes, 1 inspirational talk & 4 interactive workshops!

Discounts for Delegates!
Don't miss out on our discounts; Early Bird, Group, Academic & more!

Join us for the 20th EuroSTAR conference. See you in Amsterdam!

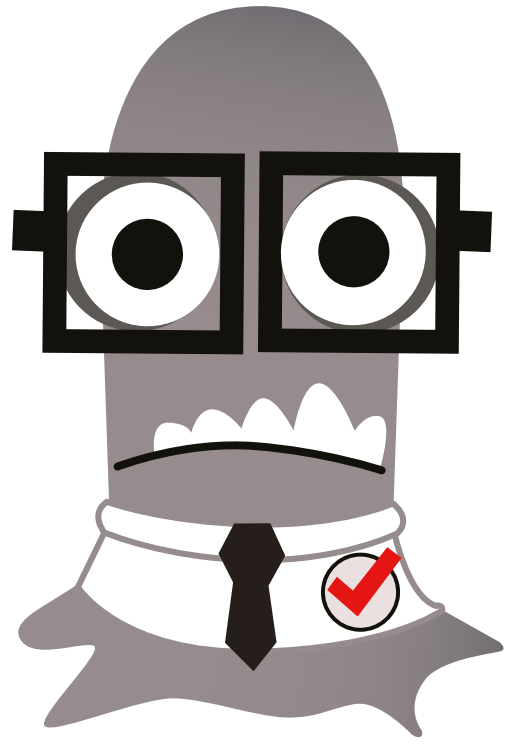
Join the EuroSTAR Community today!

Connect with fellow testers and benefit from a variety of testing resources to improve your skills and keep you up-to-date!

eBooks | Webinars | EuroSTAR TV | Blog | Podcasts

www.eurostarconferences.com

Continuous integration for testers



By Jerry Schwartz

If you want to be lean, get to the point. Software quality is often as much derived from operational efficiencies as much as it is from finding bugs. If you can build it better, and if you can construct a better workflow, you'll greatly increase the chances that the software you release will be a better experience for your end-users.

What is continuous integration? A simple definition of continuous integration is the practice of developers integrating their code with a mainline branch on a regular and frequent schedule. From this beginning we move along a spectrum of increasing sophistication, where concepts of automated builds, build on every commit, automated deployments, unit tests, and test automation come into play. An endpoint of this spectrum, though not necessarily a goal, is continuous delivery and continuous deployment, in which everything along the spectrum has been automated from build to test to deployment to a production environment. In other words, automate the building of your code so you can test and release sooner!

Good Detective Work

Let's move from the beginning of the spectrum down the line to see how at each stage continuous integration can help in a variety of testing scenarios, from a basic black-box manual tester to an exploratory tester to an automation engineer. All together it helps inform and empower what can be done. It speeds up the process and reduces technical debt. How is this accomplished?

Well, let's accept the maxim that using source material is better than using indirect documentation. If you were a detective, you would prefer to examine the crime scene in person right after it happened instead of a relying on a third-hand account from someone who saw a photograph taken a week after the event. This principle also applies to software testing. Rather than getting software built under unknown conditions days or weeks after the developer has moved on from what they have created, using automated CI the latest build is always available with the changes documented.

Getting Started

Let's say your group just wants to dip their toe in the water and asks you, the tester, to help. They decide to use a CI platform such as Jenkins, CruiseControl, MS Team Foundation Server, Bamboo, or TeamCity. For this text we'll choose Jenkins, a free open-source application that commands a large market share and supports an active community. It can take only 10 minutes to get it installed and you can run it on your own machine until you learn the basics.

Paired with one of the developers, you create your first build job. You set to pull from source-control, configure it to run their build script, and you click the shiny build now button.

Most likely, it didn't work. The application didn't get built. There are errors in the log. You make some changes and try again. Same result. You dig deeper. It becomes apparent that the build script simply wasn't designed for this. It works great on the developer's machine after some tweaking and contortions. And the developer you are paired with may exclaim, "Oh yeah, I forgot that we need to manually do A, B, and C each time we build."

While this may seem like a setback, this is actually a good thing. And it is where, as a tester, you have the opportunity to make a substantial contribution to the quality of the product. If you run into this scenario, then there are likely significant quality issues or pain points that are already being experienced at the point of production release. These may include:

- Only one developer can do the build, so they better not be out sick when we need them.
- Software built on a developer's machine may not be 'clean'.
- Unknown elements may be in play during a production build resulting in a complex, multi-step process that nobody has ever fully documented.

Standardizing the script, simplifying and automating the process so that it is repeatable and consistent can directly address many of these concerns. You don't need to be the farmer that grows the food, but you will be a much better chef who knows how to get the best out of their ingredients.

And this first, tentative step, seemingly small in scope, cracks the door just enough to see the possibilities ahead. You make another attempt at the clicking the build now button, which has been taunting since the original failure. But you and the developer did good work, and this time you find success. Even now, there are tangible benefits:

- It's easier for anyone on the team to kick off the build.
- The build is a known quantity, meaning the recipe and ingredients are defined to a common understanding.
- We can rebuild again to a predictable outcome, and there are no surprise dependencies as might be found if built off a developer's machine.
- There is now a history with dates and specific information on what went into each build
- We may have the ability to drill down into the source code, if the CI server is configured to do so.

If you ended here, then continued utilisation would justify the effort spent to this point. But like a drug, the benefits are addictive, and each step forward is incremental enough to seem enticing; every advancement yields some undiscovered pleasure.

Jenkins - A Day in the Life

Let's walk through a day in the life of Jenkins. A bit of back story first, if you hear or see the name Hudson, know that the Jenkins open-source project used to be under that name until sometime in early 2011. The Hudson Labs project is still active, but essentially the entire community has moved over to support the Jenkins project, including the creator

Continued on page 23

AUTHOR PROFILE - JERRY SCHWARTZ

Jerry Schwartz is a context-driven software tester based in Rochester, NY. He helps organize the Rochester Software Test meetup group that promotes the development of the local tester community. His latest area of interest has been discovering the interplay between exploratory testing, automation, continuous integration, and efficient process. Follow him on Twitter - @jerry_schwartz

Looking for Test Management software? Try out TestRails - bit.ly/stctrs

Continued from page 22

Kohsuke Kawaguchi.

Installation is easy. It is written in JAVA and can be run on either Windows or Linux. From download to finished install it should take about ten minutes. Once installed, it is managed via a webpage which can be initially accessed by http://localhost:8080. To make it easy for everyone to get to, the port can be changed in the jenkins.xml config file (found in the root install directory) and you can work with your IT department to give the machine a good hostname. Speaking from personal experience, it would also be a wise idea to consider how to back up the machine.

I encourage you to poke around in the settings for a bit, and to check out the available plug-ins page, found by clicking Manage Jenkins, then Manage Plugins. It lists the over 400 community contributed plugins that can be added via a click of a button. However, a word of caution; while most are outstanding, there are a few that may not be stable or reliable. Go ahead and read through the list, and install the ones that look interesting or useful. Here's a few that I've found helpful:

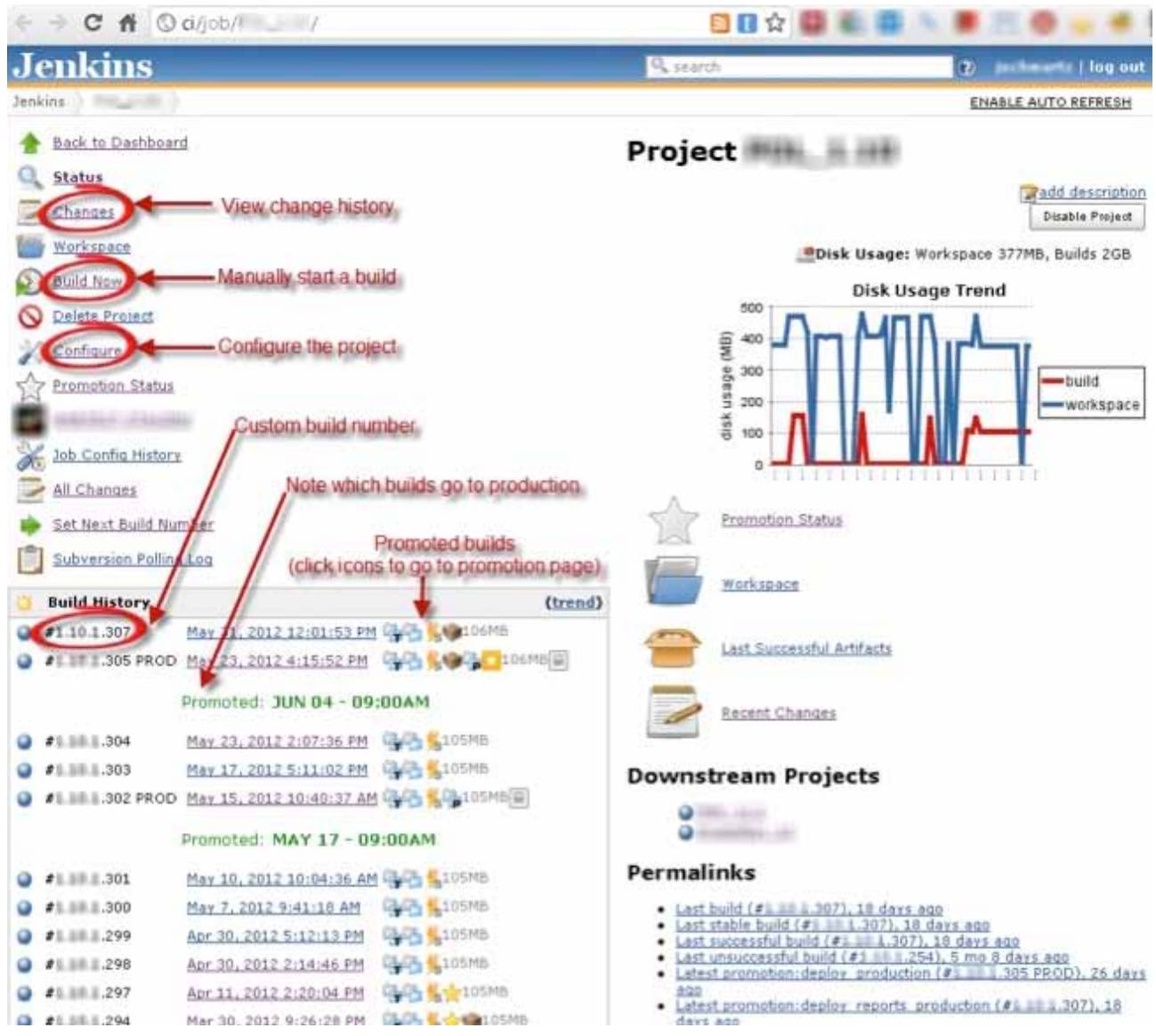
- Promoted builds: Post build options, critical if you want to create a build pipeline.
- Publish over CIFS: Copy files to windows systems.
- Email-ext: Customize the email notifications.
- Build-name-setter: Set the build number to the software version number.
- Envinject: Greater control over environment variables.
- Parameterized Trigger: Kickoff downstream jobs with parameters
- Instant-messaging and jabber notifier: Send an IM on build events.
- Subversion tagging: If you use subversion, auto-tag the build.
- ViewVC: View the source code diff.
- Active Directory: Authentication for Jenkins, other options available.

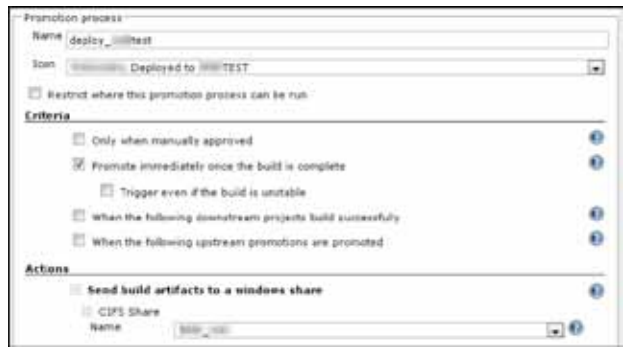
And there are many more. Odds are if your development or QA group is using a tool, there is a plugin to help integrate it into Jenkins.

When getting started for the very first time, I'd recommend starting with a few basic and obvious plugins, and creating a test job to get a feel of how things work. This way you can separate between what Jenkins offers as a base configuration and the plugin enhancements.

Your First Job

Let's get started. On the main page, click New Job. You'll be presented with several options, but for now all you should do is give it a name, chose Build a free-style software project, and click OK. Without worrying about all of the presented options, put in the source code repository link under the Source Code Management section. Now under the Build section, click Add build step and enter in one of the options that is likely to fit your environment if you were to run it on the command line. For example, on a windows system you would likely choose an





Execute Windows batch command. In the text box that is displayed, enter in the command that you or the developer may run if building manually. This is the first step of automation, and once this concept sinks in, everything else will begin to make sense.

Probably the most useful of all the plugins is the promoted build plugin. You can create and control the deployment workflow with this enhancement. For some examples, I've used it to create discrete steps to deploy (copy) to the test server and to production servers. They are also used to activate the deployment by running additional scripts that will change a virtual directory to point to the new code. Additionally, they can be used

Continued on page 24

Configuring your first job

Continued from page 23

to kick off other Jenkins jobs while passing in parameters. In this way, you can really develop a sophisticated build pipeline with upstream and downstream jobs.

A common configuration would be to have a separate job that runs the test suites, with examples including unit tests, Selenium automation and eggPlant robotic GUI testing. This keeps the build fast and gives some control over when and which test suites get run. Having a smoke test suite that runs quickly can be run more often than a full regression suite, which may be run overnight.

All these steps and additions can be added incrementally. The goal should be to just do what makes sense and what makes everyone's lives easier. My own experience was that I built up a solid pipeline without knowing the concept had a name; I simply fixed a pain point, reassessed, and fixed the next pain point. All the while I gained further insight on what the developers were creating, which in turn made my testing much more effective.

As a tester, I simply wanted to have some influence on the builds I was getting. I wanted to see what was in the build, to know its history, and to get a build without waiting or fuss. Knowing this, I could make better risk-assessments, adapt my testing plan accordingly, and cover more testing ground in a shorter amount of time. This is why I believe testers should get involved in CI. It has too much impact on software quality to cede our ground to others or be ignored. □

LEARN EVEN MORE ABOUT CI

Visit http://jenkins-ci.org and browse. It has an active community and the content is fairly comprehensive. More detailed questions and answers can be found by searching http://stackoverflow.com. To get a sense of how others use Jenkins, search Google for "Dashboard [Jenkins]" – including the quotes. Finally, to test drive Jenkins without a full local install, go here and launch through Java Web Start: https://wiki.jenkins-ci.org/display/JENKINS/Meet+Jenkins#MeetJenkins-TestDrive.

To learn more about the basics and get a deeper understanding of continuous integration, visit http://martinfowler.com/articles/continuousIntegration.html. The author, Martin Fowler, in addition to maintaining a blog where he goes expands on many of these concepts in elegant detail, also wrote one of the definitive books on this topic called "Continuous Delivery".

And for even more information on the definition of continuous delivery, visit the following page for a great explanation as well as a comparison with the meaning of continuous deployment: http://continuous-delivery.com/2010/08/continuous-delivery-vs-continuous-deployment.

The Evil Tester Question Time

Provocative advice for testers who don't know what to do!

Q1. HOLA EVIL TESTER...

How can I convince my managers to let me get involved and begin testing at the requirement stage? - Sonz

Dear Sonz,

Hmmm. There are a lot of ways to do this. The testing industry has been building up a lot of material and books and standards and processes for years that cover this very topic.

Sadly, I'm not going to encourage you with their reasons because I think that most of that material presents 'testing' as: "start spending 100% of my time on the project, formally writing a test strategy, approach, and plan, and writing test cases and scripts which are cross referenced to the requirements, even though the requirements are changing and therefore much of the 'testing' would lead to waste and re-work".

I've done this myself. I had to do a lot of rework, and saw a lot of waste. I don't want to see you do that to yourself.

Do you think your managers hear you asking to be allowed to do that? If I were your manager I would want to know what value you would add? How much time you think you need to spend? What risks there are to the project if you are not involved now? How will your involvement in this new project impact your current project? What do you think you would produce as a result of your involvement? What is the risk of re-work to the products you will produce over this time? How will it benefit the future of the project for you to be involved?

But beware. If they let you in, then you have the responsibility of demonstrating that you can add value early in the project.

And if that statement has you provoked you into a rant; "How dare he question my ability! Why I can add value easily by..." Then verbalise your rant to them.

If you can convince people that your involvement will add value, and if they are good, and if they are in control of the process, then they will let you add that value. Just make sure you avoid waste,

CUDDLY UNCLE EVIL

Q2. HELLO EVIL TESTER...

Can I make a decent living as a freelance tester? - Paul Gigi

Hi Paul,

If by freelance you mean, just randomly test things and submit bug reports and hope to be paid for them then, no, no I don't think you can. Although if you

Advertising Blooper - http://bit.ly/adblooper

figure out how to, then please let me know as I'd like to follow your example.

If you mean, work as a contract tester where someone hires you to test stuff on a contractual basis, then yes, some people do manage to make a decent living as a contractor who tests.

The best way I know to get started is to make sure your skills are honed and in demand. Market yourself effectively in your CV and create a web presence to elevate you above most of the people applying for work. Interview well and honestly, add value to the workplace when you do work.

Fear not though. Even if you currently can't do the things above there are enough ineffective hiring managers around that you can make a decent living from creating a fake CV, lying about having a bunch of in vogue certifications, and exaggerating your experience. You might not enjoy the end result though.

CAREER OFFICER EVIL

P.S. I'm not currently hiring.

Q3. DEAR EVIL TESTER...

How must a tester deal with a developer, especially when the developer carries the attitude that he is always right? - Bhavya Hegde

Hi Bhavya,

I have similar problems. Since I too am always right I occasionally butt heads with a misguided developer who thinks that they are more right than me.

If you are 100% sure that they are not right, and have evidence, then let the evidence speak for itself. Of course the evidence may have to speak to the developer's manager since the developer can always block out the evidence through clever selective listening.

Sometimes I find it useful to compromise. Let them win half the argument, wait till they've fixed half of what you want. Then start up the argument later and fight for the second half.

Sometimes I listen to the developer, and sometimes when I do that I find myself being influenced by their argument, because sometimes they are not wrong.

I've always liked these words by Fritz Perls from "Gestalt Therapy Verbatim" and when I remember them, they help me.

"I do my thing and you do your thing. I am not in this world to live up to your expectations, And you are not in this world to live up to mine. You are you, and I am I, and if by chance we find each other, it's beautiful. If not, it can't be helped."

Hope that helps,

TEAM DYNAMICS THERAPIST EVIL

Q4. DEAR MR. E. TESTER...

If Pinocchio were to say "My nose will grow now", what would happen? - James Pullar

Dear James,

Ah, philosophy involving an impertinent nose.

Fortunately for me, Pinocchio's reaction is physiological rather than philosophical. A curious thing though. Pinocchio's nose grows when he experiences cognitive dissonance and, when he knowingly and maliciously tells a lie, but... "Unluckily, in a Marionette's life there's always a BUT which is apt to spoil everything." But, not always, because we know that Pinocchio's nose does not grow when he lies if he lies because he is too embarrassed to admit the truth.

Because we testers are versed in the arts of System Thinking and Modelling, we would model Pinocchio as a complex and probabilistic system. With Pinocchio's nose as one system, having a homeostatic relationship to the system of a living wooden doll.

Also, in our reading of Pinocchio we see that he takes no pleasure in the growing of his nose and is normally embarrassed by it, so probably I think, his nose would not grow.

A counter question to you dear reader: If Pinocchio were an experienced test manager and he wrote in a test strategy "Testing will demonstrate that the system is fit for purpose to go live". Would his nose grow?

Yours, physiologically incapable of performing philosophy,

UNCLE E

Q5. HI EVIL TESTER...

What's the best way to deal with a fellow tester who is not pulling his/her weight? - Anon

Dear Anon,

The 'best' way. I don't know. My advice isn't usually 'best'. And I don't normally do manual labour so this is a very tricky question to answer. I have been a manager though. And I have seen unbalanced teams where some testers appear to be doing more work than others.

My first step is to check my observation. Iinvestigate if the person is actually under performing. Sometimes they are performing differently and the observations we are making don't include all the work they are doing. Generally I follow Deming's advice and try to change the system to help prevent such misunderstandings, or change the system so that under-performance is shown clearly.

Your question suggests that the tester in question is a peer, rather than someone you manage. So you may not be in full possession of the facts relating to your peer.

My advice to you is to raise your concerns to your manager, after all your lazy manager usually has plenty of time on their hands, and it is their responsibility to deal with your lightweight under performing co-workers.

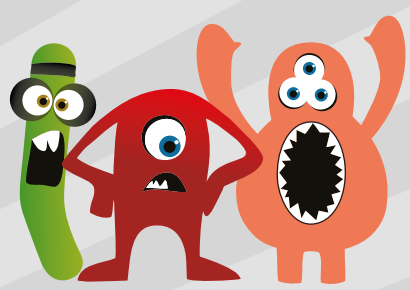
Yours,

TEAM SPIRIT COACH EVIL □

WWW.SOFTWARETESTINGCLUB.COM

Software Testing Club

a community for software testers



Coaching Testing Skills - http://bit.ly/coachingselena

NEWS IN BRIEF

A ROUND-UP OF STORIES SUBMITTED BY THE COMMUNITY.

SUBMIT YOUR NEWS HERE - HTTP://WWW.THETESTINGPLANET.COM/SUBMIT-CONTENT/

AST are working with SummerQAmp to create ideas and training materials for their initiative to train youth in IT skills. Support and suggestions are welcome from the AST team.

http://www.associationforsoftwaretesting.org/2012/06/12/update-on-summerqamp/

NIELSEN comes under fire and responds to critics on his 'backward mobile' recommendations.

http://www.netmagazine.com/interviews/nielsen-responds-mobile-criticism

---

SHARNESS is a portable shell library to write, run, and analyze automated tests for Unix programs, version 2.2 is out.

Feedback to @mlafeldt is welcome!

NEW SELENIUM Introduction course introduced by Edgewords.

Edgewords, test tools training specialist, has announced the dates of its first Selenium Introduction course. The first public course is being held in London on 2nd and 3rd July; this is a two day course that has been developed by one of our industry's leading Selenium experts; it focusses on real world problem solving and delivers practical knowledge.

Visit - http://www.edgewords.co.uk/

ISO 29119 - There's talk of a new definite standard for software testing. What are your thoughts about it?

http://www.softwaretestingclub.com/forum/topics/iso-29119-the-new-international-software-testing-standard-what

---

POTSLIGHTNING - A Low-Budget, Non-Profit, Free-Entry Peer Event.

In Germany on 17th of November 2012. Deadlines for contributions are 31st of June 2012.

The theme is:

- The role of the tester in agile;
- or why do testers have their own conference and do not go to general agile conferences
- Training and coaching
- Transition to agile – approaches, obstacles, practical experiences?
- Agile pitfalls, common reason for "failures"; lessons learned, etc.

http://gate-workshop.de/potslightning/

SOASTA'S CloudTest now runs on the HP Cloud! Cloud Testing is clearly validated as the new approach to realistic scale web and mobile testing using cloud with this news. This enables even more distributed load for users of the very popular CloudTest platform so testing at full scale is fast and easy.

http://www.businesswire.com/news/home/20120510005429/en/SOASTA-Brings-CloudTest-HP-Cloud-Services

BUGS IN THE WILD

"Technical issues" with banking systems prevents bank accounts from being updated and causes days of misery for millions of Natwest and RBS customers.

http://www.telegraph.co.uk/finance/personalfinance/consumertips/banking/9346893/Millions-of-Natwest-and-RBS-customers-left-high-and-dry.html

---

Leading comparison websites are still letting down disabled and the older generation. Their websites are barely scraping the minimum accessibility requirements.

http://www.guardian.co.uk/money/2012/apr/17/price-comparison-sites-disabled-older-people-struggle

AN INTERVIEW with James Bach by CodeCentric available online at http://www.codecentric.nl/2012/04/27/james-bach-interview/

---

30 - a Key Number for Test Effectiveness - A paper by Kalistick.

30% of tests performed are ineffective
30% of tests cover 65% of regression risks
30% of tests are redundant

Download - http://www.kalistick.com/public/white_paper/Kalistick-white-pape-2012-30_a_key_number_for_test_effectiveness.pdf

---

Continued on page 27

A Testing Toolkit - http://bit.ly/testtoolkit

Continued from page 26

TOUCHSTONE Technology Training has announced its Software Quality Assurance and Agile Development training courses in Denver.

Touchstone Technology Training offers comprehensive Software QA training and Agile Software Development courses to suit the needs of both individuals and organizations alike. The courses are designed to provide the foundation and skills to evaluate, plan, and execute effectively on Software Quality Assurance and Test activities.

Headquartered in Denver, Colorado, Touchstone Technology Training (http://www.touchstone-technologytraining.com) has conducted training for software quality assurance professionals and technology leaders throughout the U.S.

SMARTBEAR has just introduced the first commercially available version of its (once an Eviware product prior to acquisition in July '11) almost 2-year old Open Source API/Web Services load testing tool, loadUI, now offered as loadUI Pro. loadUI Pro has server monitoring. Server monitoring enables testers to easily find the solution to the performance issues loadUI finds in their APIs. loadUI Pro also can create real-time graphs of API performance and server status, and automatically saves hundreds of data points for later analysis, post test.

Full news - http://smartbear.com/news/news-releases/SmartBear-Advances-Load-Testing-for-Web-Services-a

---

FEEDBACK Sought on New Open Source Web Test Tool

Adrian Dorache recently launched a new open source web testing tool and would love your feedback!

You can download it from - http://open-twebst.codecentrix.com/ □

Keeping Your Automated Tests Lean

By Jim Holmes

Lean's principles of focusing on high-value, just-in time minimalism has gradually been spreading through the software construction domain. Lean's not just for the development aspect of our software projects, though. I'm a firm believer there's a critical need for teams working with automated tests to dive in to the Lean mindset when they're thinking about their automated suites.

Automation brings us tremendous value for the software projects we work on. We can increase that value by bringing some Lean ideas to what we test, how we test, and how we maintain our automation. When we lose sight of Lean principles, our automation suites can become risky, burdensome drags on our teams' productivity..

What We Test

Focusing your effort on the highest value work items has always been my largest takeaway from Lean. Why spend any time building something few people ever use, or that isn't mission-critical?

I live and breathe that same idea with the approach I use for automated testing on the teams I work with, especially when it comes to test automation at the UI level. UI automation is difficult to create, time-consuming to run, and can be even more of a time cost for maintaining it. If that's the case, it's critical to spend our time automating only tests that are high value.

For example, I'm a big fan of ensuring we get automation around the "Show Me The Money" test cases (phrase courtesy of Adam Goucher). By this I mean focusing on use cases that earn the company revenue or expose the organisation to liability risk. Automating tests around shopping cart use cases is one situation. Another might be wrapping critical security features with automation to ensure you're not inadvertently injecting regressions into areas protecting users' privacy information, or sensitive organizational data.

Lean test cases aren't just about what to focus writing automation for; it's every bit as much understanding what not to automate. Time and time again I see teams trying to create automation around test cases that are time intensive, brittle, and better left to manual validation.

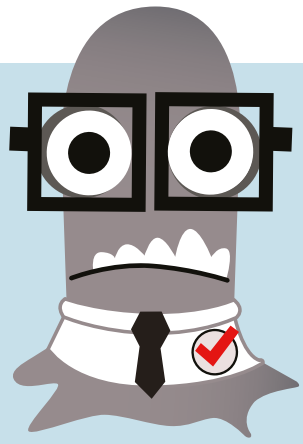
A great example of this would be comparing visual look and feel across different browser types. Trying to write this sort of automation is, quite frankly, a waste of effort in my experience. The same argument can be made for many things relating to look and feel. It's one thing to validate that a specific image is loaded in a specific element; it's another to try and validate element alignment and styling through automation.

How We Write Our Tests

Lean's minimalistic philosophy really appeals to me. I closely associate this with the "YAGNI" or "You Ain't Gonna Need It" outlook for building software. Lean and YAGNI both push you to avoid building things you THINK you might need later and instead focus on things you KNOW you need right now.

Using this same approach with automated tests helps us ensure we're only doing work that's of the highest value. Focusing on high value test cases makes us write tests that validate business-critical features like realising revenue or security infrastructure if your system holds information

Continued on page 28



TESTING TIPS

Do you do web testing?
Perhaps you should understand
how browsers work -
http://www.html5rocks.com/en/tutorials/internals/howbrowserswork/

About 8% of the male population have some kind of colour blindness.
Have you considered how colourblindness can effect the user experience?
http://wearecolorblind.com/

Like to think visually?
Then perhaps try MindMapping.
Here are some testing related MindMaps -
http://pinterest.com/rosiesherry/testing-mindmaps/

Would you like to learn to code?
Check out Codecademy -
http://www.codecademy.com/

Automation suites can become a burden on team productivity

Continued from page 27

covered by various regulatory enforcement policies. (Think financials covered by SOX compliance, or patient data covered by HIPA .)

If we're thinking Lean, then we don't spend time on tests that aren't a good target for automation such as testing styling, layout, or cross-browser rendering equality. Those tests are usually extremely difficult to write and worse to maintain over time. Instead, leave those sorts of tests for manual validation.

Lean in test automation isn't just about the large-scale business value; it's also about the small details of the tests themselves. For example, if you're using the page object pattern to layout your tests (and you should be!), then there's no need to go overboard when starting to build out your test suite.

Do not jump in to creating a new page object for every page in your system. Build page objects only for the high-value tests you're working on right now. Do not create properties or accessors for every field and element on the page you're working with. Instead, focus on mapping out only the services and elements you need for the current test.

Staying Lean

In my view, Lean focuses not just on building high-value features, but also on maintaining only high-value features in your system. If a feature is being used by only a fraction of your user base, then that feature is a ripe candidate for removal. Why spend the time and energy to continue maintaining that feature when you could simply prune it, and invest that effort elsewhere in your system?

The same concept applies to our automation test suites. Test automation, particularly at the UI level and somewhat less so at integration and unit levels, can be long running and somewhat expensive to maintain. Why not be just as aggressive about pruning out low-value or expensive-to-maintain, tetchy automation?

At my previous job I was responsible for a team that was running over 9,000 tests in approximately 900 Selenium fixtures. That team actually wrote close to 15,000 Selenium/WebDriver tests over the space of two years; however, we were extremely aggressive about monitoring our test coverage and dropping fixtures or tests that had become outdated or less valuable. This helped ensure we were testing the right things, and not wasting time maintaining low-value tests.

Migration of tests is another topic that's a great example of keeping Lean values close to your heart. Just because your team or organisation finds a new test tool or framework doesn't mean you should jump in to rewriting all your existing tests with the new toolset. Migrating entire test suites is rarely a good use of time and effort. Instead, focus only on high-value tests.

Your existing tests are (hopefully!) providing great value to your projects. Why touch them? Leave them exactly as they are, and focus on writing new tests in your new toolset! If one of the tests in the older toolset becomes outdated, drop it from that environment. If one of the old tests is valuable, but breaks, re-write it in your new toolset.

Lean Isn't Just for Systems

Lean helps us focus on delivering great value to our customers, regardless of who they are. Minimalism, clarity, and reducing waste are huge benefits of a Lean mindset. Why wouldn't we want to apply those same ideas to our test automation software too? □

AUTHOR PROFILE - JIM HOLMES

Father. Husband. Geek. Veteran. Around 25 years IT experience. Co-author of "Windows Developer Power Tools." Coffee Roaster. MVP for C#. Chief Cat Herder of the Code-Mash Conference. Diabetic. Runner. Liked 5th grade so much he did it twice. One-time setter, middle blocker, and weakside hitter. Blogger (http://FrazzledDad.com). Evangelist for Telerik's Test Studio, an awesome set of tools to help teams deliver better software. Big fan of naps.

NEW BOOKS

THE SOFTWARE MINEFIELD
By Mike Talks

http://leanpub.com/The-SoftwareMinefield

HOW GOOGLE TESTS SOFTWARE
By James A. Whittaker, Jason Arbon and Jeff Carollo

http://amzn.to/hgtsbook

EXPERIENCES OF TEST AUTOMATION
By Dorothy Graham and Mark Fewster

http://amzn.to/eotabook

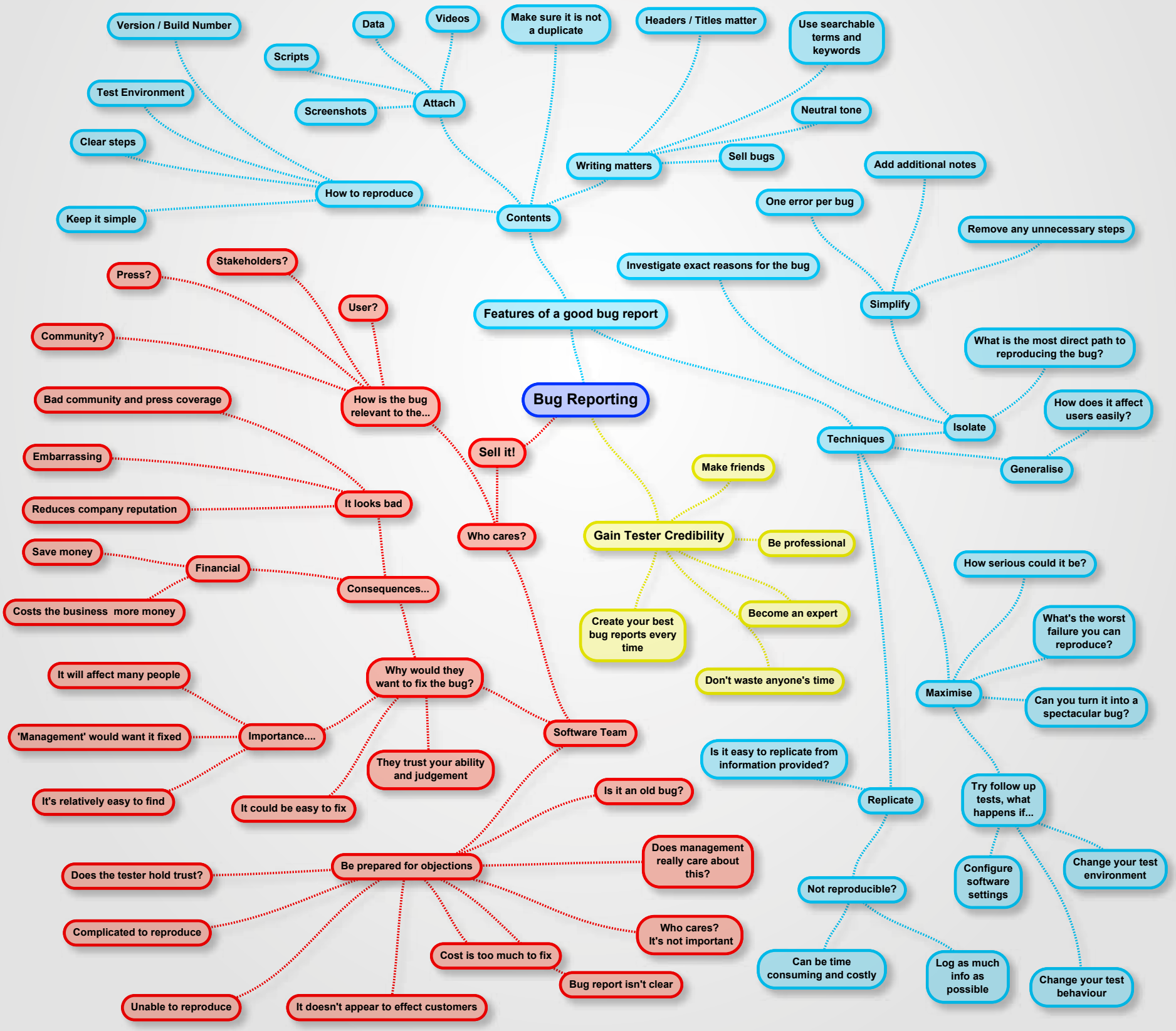
THE LEAN STARTUP
By Eric Ries

http://amzn.to/leanstartupttp

ALWAYS A DUCK
By Elisabeth Hendrickson

http://leanpub.com/alwaysaduck

BUG REPORTING MINDMAP



A picture is worth a 1000 words - http://bit.ly/1000picture

THE TESTING PLANET DIRECTORY - GET LISTED WITH THESE AWESOME COMPANIES - thetestingplanet.com/directory

TEST TOOLS & SOFTWARE

GEMINI
Gemini brings versatile test management, bug and issue tracking to your team. Sign up to our cloud-based offering or install locally. Join the new generation in software project management with Gemini – no hidden extras or crazy pricing. 3 Users FREE – No Gimmicks – Full Edition. **www.geminiplatform.com**

TESTRAIL
TestRail – Test Case Management Software for QA and Development Teams. Comprehensive web-based test case management software to efficiently manage, track and organize your software testing efforts. **www.gurock.com/testrail**

PRACTITEST
Practitest is a SaaS-based Test Management Solution that supports the entire QA lifecycle, including requirement & issue tracking. **www.practitest.com**

REQTEST
ReQtest is an easy to use bug tracking software, available in the cloud 24/7. It empowers teams to work more efficiently and gives decision makers meaningful data on progress made. ReQtest includes a requirement management module which is tightly integrated with the bug tracking features. **www.reqtest.com**

KALISTICK
Kalistick gives testers a new solution to design efficient test strategies focusing on business risks. Our unique technology analyzes test cases footprints and functional changes to select the most relevant test cases. Discover how to move one step ahead in testing efficiency. **www.kalistick.com**

BUG DIGGER
BugDigger removes the hard work from web site bug reporting. With the help of a browser add-on, automatically captures and uploads: – web page screendump optionally annotated using built-in editor, – environment details, and – web site usage history. Even busy or inexpert testers can create useful bug reports instantly. BugDigger integrates with JIRA, Basecamp, Pivotal Tracker, FogBugz, Unfuddle, Redmine and others. **www.bugdigger.com**

TESTWAVE
TestWave is a next generation test management tool implemented as Software as a Service (SaaS). It can be deployed instantly and you only pay for what you use. TestWave is designed for both Test Managers and Testers, and provides requirements, test planning, test execution and defect tracking. Intuitive graphs report testing data in real time. Reduce your costs and unleash the power of SaaS with the cloud's first fully extensible test management tool. Learn more and sign up for a free 30 day evaluation: **www.testwave.co.uk**

PARASOFT SOATEST
Parasoft SOAtest automates web application testing, message/protocol testing, cloud testing and security testing. Parasoft SOAtest and Parasoft Load Test (packaged together) ensure secure, reliable, compliant business processes and seamlessly integrate with Parasoft language products (e.g., Parasoft Jtest) to help teams prevent and detect application-layer defects from the start of the SDLC. Moreover, Parasoft SOAtest integrates with Parasoft Virtualize to provide comprehensive access to traditionally difficult or expensive to access development and test environments. Parasoft SOAtest provides an integrated solution for: End-to-end testing, Environment management, Quality governance, Process visibility and control. **www.parasoft.com**

TESTPLANT
TestPlant develops eggPlant the leading user interface test tool that creates an abstraction of a GUI for any device type, enabling automation of screen-based testing through 'search and compare'. Download now. **www.testplant.com**

XSTUDIO
XStudio is a free ALM/test management solution allowing to manage requirements/specifications, scrum projects, Automated/manual tests, campaigns and defects. An LGPL SDK is also included to interface with proprietary tests. **www.xqual.com**

TESTLODGE
TestLodge is an online test case management tool that allows you to manage your test plans, requirements, test cases and test runs with ease along with issue tracker integration. **www.testlodge.com**

TESTOPTIMAL
TestOptimal – Model-based data-driven test design and test automation to improve test coverage, enable rapid response to changes and reduce test maintenance cost. **www.testoptimal.com**

LOADSTORM
LoadStorm – The lowest cost and easiest cloud load testing tool. Free account for 25 users. Test up to 100k vusers. Real-time graphs with key performance metrics. **www.loadstorm.com**

SOFTWARE TESTING TRAINING

SKILLS MATTER
Skills Matter supports a community of 35,000 Software Professionals with the learning and sharing of skills to write better software. Find hundreds of meetups, talks, conferences, skillscasts and workshops on our website: **www.skillsmatter.com**

THE TESTING PLANET DIRECTORY - GET LISTED WITH THESE AWESOME COMPANIES - thetestingplanet.com/directory

An eight level model for exploratory testing - http://bit.ly/8layers

THE TESTING PLANET DIRECTORY - GET LISTED WITH THESE AWESOME COMPANIES - thetestingplanet.com/directory

SOFTWARE TESTING SOLUTIONS

ElectroMind
YOUR SOFTWARE QUALITY MATTERS

ELECTROMIND

ElectroMind offers training, consulting, coaching and mentoring services to the software testing community. Through strong relationships with world-class testing experts, built up over several years, ElectroMind delivers niche training products, test process improvement consultancy and innovative people skills development programmes. Our consultants are comfortable using both traditional and Agile testing methodologies with experience in several industry sectors including financial services, telecommunications, online retail, travel, mobile and digital media. Through strategic partners, ElectroMind can offer performance engineering services including load and stress testing. Our overall philosophy is simple. We believe your software quality matters. **www.electromind.com**

Test Hats
Software Testing

TEST HATS

Test Hats are an independent software testing services provider, with offices in the UK and Spain. We provide a full range of testing services including System, Performance and Security testing along with specialised Consultancy and Training. For near-shore testing our Test Lab is fully equipped with a range of desktop and mobile platforms, testing software and tools, allowing us to provide a quality service at a competitive price. Visit our website to learn more about Test Hats and our services. Get in touch today to talk about how we can help test your projects. **www.testhats.com**

The TestPeople

THE TEST PEOPLE

The Test People delivers the best, most innovative, highly technical and competitive performance engineering and test service available today. Based upon our extensive experience, TTP can deliver tailored services to address all aspects of the functional and non-functional test lifecycle, including highly specialised performance engineering and test automation services including automated build and continuous integration solutions. TTP are at the forefront of utilising the cloud for test and load environments, with significant experience in open source and the major commercial toolsets whilst also coming armed with our own performance and automation frameworks. **www.thetestpeople.com**

ORIGINAL SOFTWARE

Original Software - With a world class record of innovation, Original Software offers a solution focused completely on the goal of effective quality management. By embracing the full spectrum of Application Quality Management across a wide range of applications and environments, products include a quality management platform, dynamic manual testing, robust test automation and test data management. More than 400 organisations operating in over 30 countries use Original Software solutions. Amongst its customers are Coca-Cola, Unilever, Barclays Bank, HSBC, FedEx, Pfizer, DHL and many others. Visit **www.origsoft.com/solutions** for more information.

Moolya <testing>
brainual software testing services

MOOLYA

Moolya is a new generation software testing services company headquartered in Bangalore, India founded in 2010. Our focus is to help business move forward. We believe in helping our clients to make great products that wow their customers. That's when we win. We are context driven testers highly skilled at exploratory testing, SBTM, small "a" agile testing and check automation. How can we help you win smiles on your customer's face? sales@moolya.com / **www.moolya.com**

REVOLUTION IT

REVOLUTION IT

Revolution IT is the leading Quality Assurance and Testing, management consulting firm in Asia Pacific. We help our clients deliver IT projects and have core offerings across Project Management, Requirements Management and Application Testing. We have over 250 staff and offices in Melbourne, Sydney, Brisbane, Canberra, Adelaide and Singapore. Our offering includes delivery consulting, methodologies, tool solutions and training. We have strategic partnerships with HP software, IBM Rational, Oracle, Agile Academy and SAP. With HP we have been the leading HP Software Platinum Partner for 4 years running and the leading reseller, 1st line technical support, training and services partner. **www.revolutionit.com.au**

INDEPENDENT TESTERS, TRAINERS AND CONSULTANTS

testingtimes
INDEPENDENTSOFTWARETESTING

ANNE-MARIE CHARRETT

Anne-Marie Charrett is a testing coach and trainer with a passion for helping testers discover their testing strengths and become the testers they aspire to be. She offers a blend of online coaching and training on Exploratory Testing, Career Management and motivating testers. Anne-Marie is currently working on a coaching testers book with James Bach, due out late next year. **www.testingtimes.com.au**

COMMUNITIES, CONFERENCES AND NEWS

EuroSTAR

EUROSTAR

EuroSTAR is Europe's premier software testing event and will be taking place this year in Manchester, UK from November 21 – November 24. At EuroSTAR 2011, the leading names in testing will meet for an intensive 3-4 days of learning, networking, discussion...and a few extracurricular activities! Attendees can choose from numerous thought-provoking presentations, intensive tutorials, interactive sessions and inspirational keynotes. Plus, visit Europe's largest software testing exhibition which will be showcasing the leading companies in the industry. The EuroSTAR Team hopes to see you in Manchester later this year for what will be a fantastic, fun and innovative conference! **www.eurostarconferences.com** □

THE TESTING PLANET DIRECTORY - GET LISTED WITH THESE AWESOME COMPANIES - thetestingplanet.com/directory

The Evil Tester's Guide to Evil - http://bit.ly/eviltester


If there's a software problem, we'll find it.
With complex software code, the smallest of errors often cause the biggest headaches. At MagenTys, we've established a successful track record for delivering the most rigorous bespoke testing solutions to blue chip companies for complex and business critical IT programmes. The results have saved time, money and reputations. To discover how MagenTys can help you and your organisation reduce project risks, call us on 0208 667 2101 and quote 'Times 2012' to receive a discounted Test Practice Health Check. Alternatively visit magentys.co.uk
MagenTys™
The software testing experts