The trapeze is an aerial apparatus commonly used in circus performance. It makes me think of flight; two acrobats, swinging in synchronisation, then a well-timed leap of faith. The perfect circus act for which to name our new magazine; reflecting our association with Testing Circus and our goal to share the experiences of high-flying testers, bridging the Tasman Sea.

This corner of the world is home to a remarkable community of testers who study their craft and think critically about what they do. What we lack in numbers, we make up for in enthusiasm. The response to the inception of Testing Trapeze has been supportive and encouraging. We hope that you feel proud of our magazine; that you will want to read it, share it and contribute to it.

In each edition we plan to feature four Australasian voices and one international contributor. Our first set of articles come from those prepared to take a risk on the unknown and deliver quality writing in a tight timeframe. Thank you to Aaron Hodder, Adam Howard, David Greenlees, James Bach, Kim Engel and Viktoria Kuznetcova. I appreciate your patience with a new review process and a novice editor.

Finally, as editor, I present the public face of a team. Thank you to those who have helped create our first issue. Without you, there is no magazine.

Katrina Clokie is a tester from Wellington, New Zealand. She is co-founder and organiser of WeTest Workshops, a regular participant of KWST, and the editor of Testing Trapeze. She blogs and tweets about testing.
IN MY EXPERIENCE, USABILITY TESTING IS not given the focus it deserves. This is particularly prevalent when it comes to estimations of effort. I have often been faced with products that would benefit greatly from usability testing however have not been allowed the time to undertake it using a more planned and formalised approach.

With this in mind I have developed a ‘quick’ list of usability testing tips that can assist if you too are faced with these, all too common, time poor situations.

Consistency
This is very simple and quick to assess, but is very important for usability. Have you ever navigated a website where the text was in various fonts and sizes; where some lists had bullet points and others had icons; where the colour scheme changed from page to page? Did you remain on that site for any great length of time? No? Nor did I; I tend to leave the website as quickly as possible. As a user these types of inconsistency issues paint a picture of un-professionalism. How good are their products and/or services going to be if they cannot even present a consistent website?

Even the quickest of scans over the pages of a website can uncover easy to resolve inconsistencies. It’s surprising how quickly these issues stand out if you give them your focus for just a few moments.
Browse

Would you consider the website 'easy' to navigate? Would your potential users consider the website 'easy' to navigate? The latter question is a difficult one to answer, however you can use your answer to the former question to help. If you answered that the website was easy for you to navigate, then think about why that is. Do you have any particular experience that has allowed you to answer in the positive? Is it likely that the website’s potential users will share that type of experience?

One thing that does assist in navigation is breadcrumbs. The use of 'traditional' breadcrumbs is reducing; however I still believe these are a great idea, and they can be done to match modern themes. Besides, for users that don’t need them they really do no harm at all as their required screen real estate is minimal. One of the most important aspects of navigation is that you don’t want users to have to follow the same path several times in order to get to different locations. Make sure each of the important sections of your site are accessible within a few clicks from the homepage.

Also, be aware of Dark Patterns, a type of user interface that has been carefully developed in order to trick users into taking certain actions. These are very common on installation programs whereby the distributor tricks the user into installing other applications via clever wording and placement of the ‘agree’ and ‘disagree’ buttons. There is an organisation out there willing to name and shame, so please take the time to familiarise yourself with some of the more common approaches.

Search

Does the website have a search function? Is it easy to locate? Does it search the whole site, or only particular sections? Do you need it to search only in particular sections? All good questions to ask yourself while testing.

After that you can check the search results. While the relevance of the results may be more of a functional aspect of the website, for usability purposes you can focus on how they are presented. On many sites the search results return lengthy text and URLs that really don't make much sense at first glance. I find that it helps to have the headings of each
section of the website where your search results reside as it allows the user a greater chance of selecting the most appropriate result the first time. An example of this is illustrated above.

**Look & Feel**

This is broad, however it’s broad because it’s a big deal. I hear it often; “Usability is about the look and feel of the product.” The critical tester in me wants to scream and shout, “No! It’s much more than that!” And while you’d think I was correct; perhaps I’m not. ‘Look’ could account for anything visual, and ‘Feel’ could account for anything physical and emotional. It does help to break these down of course, but for the purpose of a quick usability assessment think about your initial reaction to the overall look and feel of the website you’re testing. What is the first thing that comes to mind when you see and use the website for the first time. I often visit new websites when researching tools.
As an example, compare the two sites below and think about which one pleases you more upon first visit?

Granted, the first has a lot more to offer in terms of products, but it’s completely overwhelming in relation to content. There is simply too much for my eyes to handle. The second is clean, modern, minimalistic, and tells me straight up what it’s all about. It’s practically begging me to surf more and discover.

**Links (or not)**

While broken links are a cause of great user frustration, the focus of those would sit more squarely in the functional basket. The type of links that I like to look for when conducting a usability assessment are the ones that are not links at all. Text that is highlighted and underlined but is not actually a link. A definite no-no in the world of usability, in my humble opinion, and definitely not consistent with common approaches...
to web design. There are other ways that important text can be highlighted.

**Help/Contact Us**

This area requires a little more fore-thought than some of the others. Does the organisation behind the website want the users contacting them? Your first answer may be yes. However, I have worked for organisations where they didn’t. For example, online services via a website are a good way of reducing the need for staff in front offices due to customers being able to interact online instead, and can significantly reduce bottom line expenditure.

So with that in mind, assess the help content and contact us prompts of the website. If you want the users to do everything themselves and limit the amount of interaction required then make sure the help content meets that need. If you want to drive contact from your users, then make that as easy as possible for them. Does the site provide a web form for the user to complete; easy to find contact details; perhaps even a promise of a certain turnaround time for contact? That last point isn’t really in the realms of usability, however it should only be there if the company can honour it (think holistic user experience).

**Where are you and why?**

At any particular point while browsing a website you should be able to stop and work out exactly where you are and how you got there, if you weren’t already aware. Getting lost on a website can be very frustrating for users and can lead to premature closures of the site. Some users will get lost no matter what you do in this area, so make sure there is an easy way for them to go back to the beginning; i.e. an easy to spot link to the website’s homepage, or breadcrumbs as mentioned above.

**Content**

Why reinvent the wheel when there is such a great checklist for usable content over at [UX Booth](#)? One thing I will mention is the need to reduce the amount of content to only what is required. Steve Krug, the author of [Don’t Make Me Think](#), has a wonderful quote in relation to this; "Get rid of half the words on each page, then get rid of half of what’s left." Keep it simple and to the point; your users will appreciate the time and effort saved.
Errors
While conducting a quick assessment I wouldn't suggest invoking every possible error, unless of course that is of particular importance. However, if you do invoke these while conducting other parts of the assessment take a little time to read them. Are they helpful, do they tell the user what they did wrong in easy to understand language, do they point the user in the right direction, i.e. what they should do to correct the error?

Customisable
Moving towards the often forgotten realm of accessibility, are there any customisations available on the website? Or more importantly, should there be? A common customisation is the size of the font. This is separate to your browsers CTRL+/- functionality which enlarges or shrinks the overall resolution of the page.

Last, but definitely not least, you need to do all of the above while thinking about the...

Users
One of the first 'rules' of good communication is to tailor it to your audience. The same can be said for usability; tailor it to the users. While this sounds easy, it's not. Sure, if you have one user for your website it may not be so hard... but that scenario is not likely to occur anytime soon. The study of your website's user demographic could be a very large scale piece of work. Simply keep the users of your website in the back of your mind while conducting the above testing. Perhaps even adopt some personas while testing, which may assist in your quest for usability related issues.

The above tips are intended to be used as a guide when you need to quickly assess the usability of a website; however some of them can also be used other types of products. Please don't rely on these being all that you need, they will simply help you and your website along on the journey that is usability.

David Greenlees has been testing software for over ten years. Many of these spent in one of Australia's largest government departments, while more recently undertaking a consultant role in multiple organisations.

He is a vocal and valued member of the Context-Driven Testing Community and is extremely passionate about the betterment of the software testing craft. He has published several articles, and blogs regularly at Martial Tester and Hello Test World. In 2012, David founded the Australian Workshop on Software Testing, Australia's only Peer Conference.

Currently authoring a book on the subject of software testing and martial arts, his passion outside of work. You can follow David on Twitter via @MartialTester and @Useology.
Testing Circus is one of the world’s leading English language magazines for software testers and test enthusiasts.

Published regularly since September 2010, it has established itself as a platform for knowledge sharing in software testing profession.

Testing Circus is read by thousands of software testers worldwide.

Free monthly subscription is available at: http://www.testingcircus.com/subscribe/

FIND TESTING CIRCUS ONLINE:
www.testingcircus.com
facebook.com/testingcircus
twitter.com/testingcircus
I HAVE A TENDENCY TO BE QUITE HELPFUL. That may sound like I’m boasting, because surely helping others is a good thing, right? I do find that approaching software projects with a positive and helpful attitude is rewarding, but I’ve recently learnt that a test manager taking on too much responsibility within their team is detrimental to the quality of the release.

Upon joining a project, I personally take on a strong sense of responsibility for releasing a quality product to customers. When I investigate different angles of risks to quality, I need to know more about the processes that go into developing and releasing software at that company. As I speak with other managers about their team’s role in software product releases, we always identify multiple major project issues.

There are two issues in particular that I’ve encountered across multiple companies:

- Automated unit test results have not been checked for weeks or months
- Releases for testing and production are created by different teams.

These issues both present serious risks to product quality, and should not be ignored by the test manager. I have worked with some outstanding colleagues in the software industry; they were concerned about the risks inherent with each situation, but had limited time, resources and budget to spare for implementing solutions. Each time, I agreed (and often volunteered) to take on these responsibilities within the test team rather than accept the status quo.

Test team checking unit test results

I take my hat off to development teams who have an automated build process, with unit tests that are run as part of creating a build. Unless they never check the results of those unit tests, in which case, why even bother to have them? Too often the reason that development teams stop checking the results is that many of the unit tests have started failing! The failed tests can be time consuming to investigate, and sometimes the problem is an out-of-date test rather than a bug in the program. So development managers lower the priority of unit tests and focus on writing new
features and addressing raised defects in order to meet product deadlines.

My response to this situation was to have a tester check the unit test log each morning, and raise a defect in the defect tracking system for each failed unit test. These defects were then assigned out by the development manager and prioritised above other tasks. The problem was no longer so easy to ignore, and the unit tests were getting fixed. Yet in hindsight, I believe this approach was actually harmful to product quality.

The extra handling of defects through the tracking system was substantial. These did not go through triage, but still took time to raise, assign, re-assign, update the status, etc. All of that time spent on defect management could have been used more effectively to improve product quality.

Many developers took on less responsibility and accountability for updating unit tests while they were making code changes. They knew that if a unit test failed as a result of their change the test team would raise a defect. So they could “save time” by only updating tests once defects were assigned to them. This created a situation where some unit tests were reporting false positive results, as only the failing unit tests were being reviewed. It took a lot more time for the test team to manually detect, raise and retest bugs which should have been caught by unit tests as part of the build process.

There were some proactive developers who had already fixed their unit tests before the defect appeared in their queue. They saw these superfluous defects as a nuisance, a distraction and a waste of everybody’s time. Overall I believe that their opinion of the test team was lowered, as they saw us performing what was basically a redundant data entry function. Development and test teams function together much more productively when they have mutual respect.

I have tried a much less helpful approach on another project. I was strict about demanding that all unit tests were run and passed on a particular build prior to accepting that build for testing. This approach forced the development manager to explain every anomaly. It was damaging to the work relationship, as it put the development manager on the defensive. In practice, tests can fail or be skipped for many
different reasons, and this should be left to the discretion of the development manager.

Next time I will encourage the development manager to resolve this issue by checking unit test results within their team. That will allow developers to retain responsibility for the integrity of the unit tests. Also, developers can fix the tests and verify them without raising defects, saving on defect management overhead. For my part, I will regularly confirm that the unit test results are being followed up by the development manager, and keep the lines of communication open between the teams.

An overly helpful approach can have the unintended side effect of harming product quality in various ways. When offering to assist other teams, ensure that you discuss who will ultimately take responsibility for each aspect of the task. Also consider whether the time required for the task could be better spent on testing activities, in the interests of product quality.

**Test team creating release packages**

In another role, I was responsible for multiple projects being developed concurrently, and the sole release manager resigned shortly before one of those projects was due to be released. That person had been in the role for a long time and there was no documented release process. Without enough time to hire a replacement, I volunteered to create the release package. My theory was that I already knew the version of software to be released, and the location of the files, so how hard could it be to publish the release?

Getting access to the release management tool and learning to use it took me more than two days. It was clunky software at the time and had its own set of bugs to contend with. I found it interesting for the first hour or so, and annoying for the rest of the time. Every hour spent on this task was time taken away from test management for the concurrent software projects.

I published the release to staging, and had the test team verify it. Then I made the release publicly available, and again asked the test team to verify it before sending out release notifications. Once that was complete, the support team installed the public version in their own environment. They immediately reported that their diagnostic tools were...
missing from the release. Investigation of customer issues would be severely hampered without these tools installed and running on the customer’s computers. A little too late, I learnt that the automated build process produced more than one set of installers, in different shared folders at different network locations. The test team had been installing and testing a subset of the final product.

Looking back, I realise that a short meeting with the managers of various teams would have allowed me to create a checklist for releasing the product. As a management group, we could have then assigned out tasks across the various teams more appropriately. For example, there were developers in the same office who had experience with the release management tool. They could have made the product live in one hour rather than two days. My goal was to save the other managers some time, but the product took longer to release. Being overly helpful had a definite negative effect on the product quality of that particular release.

This was very useful information for future releases, but having the test team publish a release and also verify the release process was another case of being too helpful and negatively impacting product quality. If the test team releases the product, who tests the test team? In the absence of a release manager, it would have made more sense for the support manager to package and publish the release.

**How helpful is too helpful?**

There’s a line to be drawn between helping other teams, and taking on responsibility for aspects of that team’s role. Many times developers have helped my test teams, for example, by writing custom tools for automating specific aspects of testing. That help was much appreciated, and saved the teams an enormous amount of time and budget! Imagine if those developers had then been even more “helpful”, and had taken on the role of setting up the test scenarios, checking the results and raising defects without involvement and oversight from the test team. Certain things would either be tested by both teams or not at all, and either way there would be a negative impact on product quality.

The test manager is in a position to notice issues in the software development process, but that doesn’t automatically mean we’re best placed to resolve those issues. When offering to help, solicit input and
feedback from representatives of the relevant teams. Make sure that your reporting of those additional activities is just as transparent as your reporting of testing activities, to allow for effective feedback. And overall, consider whether the issue presents a risk to the project which is greater than the risk of having fewer resources focused on testing.

Kim Engel is a software test manager focused on user experience and fostering communication between stakeholders.

She is a regular attendee of the OZWST peer conference, an avid reader and occasional writer of testing blogs, and an infrequent tweeter @kengel100.

Kim is in the process of overcoming 10+ years of traditional testing experience to adopt a CDT approach. She is employed as a Test Consultant with Planit Software Testing Ltd in Auckland, New Zealand.
CITCON (Continuous Integration and Testing Conference) is happening in Auckland this February. CITCON brings together people from across the software development industry to discuss continuous delivery and the practices, continuous integration and testing, that go along with it.

What makes CITCON different is the unique Open Space format: attendees don’t sit in rows listening to presentations. Instead they openly discuss the topics that matter to them, with the experts, both local and visiting.

The topics are as varied as the audience. Attendees include developer-testers, tester-developers, devops and other people looking for cross-functional solutions to make software better!

CITCON is brought to you by the Open Information Foundation, co-founded by Jeffrey Fredrick and Paul Julius. Entry is free; attendance is limited to 150 delegates.

CITCON Auckland has sold out. To add your name to the waiting list, and stay informed of future events, visit the [website](#).
HERE’S A TESTING CHALLENGE FOR YOU: teach someone to test in three days.

This was the task handed to Katrina Clokie, Aaron Hodder, Georgia Chunn and myself in January. We were tasked with revamping a three day portion of Assurity’s graduate training programme which has introduced dozens of graduates to the world of testing.

This was a hefty responsibility, and we were appropriately anxious about it. After all, this would be the first introduction to testing that these ten graduates had ever had. While some had studied computer science, most hadn’t, and so they were entirely new not just to testing, but to software development and IT in general.

Thankfully, the three days went off with barely a hitch. The graduates seemed to enjoy the experience, we certainly had fun and, best of all, by the time we were done with them, our flock actually seemed to know a little something about testing. So how did we do it?

A Voyage of Discovery

Faced with the prospect of trying to impart decade’s worth of wisdom and debate to our fresh-faced and eager graduates, it would have been easy indeed for us to simply use the meagre allocation of three days we had to throw as much information as possible at them. Saturate them with ideas and hope that as many as possible stick.

After all, who are we to say what challenges they will face in the immediacy of their testing futures? While they will be allowed to develop and grow as testers in a supportive environment for the
early part of their careers, each will likely go in different directions and face a range of experiences, situations and problems.

So there could certainly be an argument for arming our graduates with as wide an array of knowledge as possible, to best prepare them for the many and varied challenges they would, inevitably, face.

However, this would have meant subjecting them to a hectic, relentless stream of information. The sort of learning experience which, rather than enlightening you, rather highlights the vast swathes of knowledge you don’t possess. It would have been constant but tedious, informative but confusing, learning but not understanding.

As such, we elected to take a different approach.

Instead of telling them what testing is and why we do it; instead of telling them about test design and analysis, or oracles and heuristics; instead of telling them about test cases, test ideas and exploratory testing; and instead of telling them about bug reporting and testing strategies - instead of all of that, we told them nothing.

Or as close to nothing as we could get away with.

To start, we asked them what they understood testing to be. They shared their understanding with each other and with us, and then we gave them something to test. Before we’d even attempted to teach them anything, they were finding things out for themselves.

And this set the precedent for the days ahead. Before we opened up and gave them any sort of information - because of course we had to tell them something about testing! - we would ask the graduates to think about it and do it for themselves first. After all testing is a cognitive, practical profession. So we wanted them to be thinking and we wanted them to be doing.

Instead of telling them, we asked them. We asked them to tell us what they thought about testing. And we asked them to actually do some testing, hands on, with real software.

It was in this way that we approached this teaching challenge. We didn’t try to teach them at all. Instead, we recognised that the best use of the limited time we had available was not to fruitlessly try to teach them everything but to engage them sufficiently such that they would
They didn’t learn testing during their time with us. They discovered it. 

A Bountiful Shore

Of course, the success of any such voyage depends upon what you find on the shores of your newly discovered land. Thankfully, it appears that our graduates’ journey of discovery immediately began to bear fruit. The outcomes from this were twofold.

Firstly, it meant that to facilitate the graduates’ learning about testing, we got them to do some testing. This was true not just in the literal sense - when we gave them practical testing challenges - but also on a more fundamental level: testing is about eliciting and communicating information, and this is precisely what we asked our students to do.

In other words, to teach testing, we did testing. This was our primary goal for our time with the graduates. Many testing courses attempt to distil a practical, active discipline into passive, theoretical knowledge. We didn’t want to just give our graduates this knowledge, we wanted to give them experience.
After all, these were graduates with no prior experience in testing. They are entering a new and intimidating world and they know that they will soon be asked to actually do some testing. When that time comes, it’ll feel good to have done some before.

And when we did challenge them to do some testing, they really stepped up. After only two days we gave them two hours to develop a strategy for a complex - and highly pressured - testing problem. We were blown away with the amount of thought, creativity and depth that went into their responses. Truly, they had discovered testing.

The second outcome which benefitted the graduates was that, by its very nature, discovery is about finding new things. For the graduates, testing was this new thing, and we wanted them to go off and discover it for themselves. We wanted to harness that excitement of coming across something new and revelling in the insight or understanding it can give you.

But we also wanted to stress that they are not the only ones discovering testing. We wanted to communicate the idea to them that testers who have been testing for years are still discovering new ideas that are relevant and crucial to the evolution and development of our craft.

To achieve this, we had to do more than just question our graduates. We also encouraged them to question us, each other and the resources we gave them. We presented them with definitions - often stamped with an acronymic seal of officialdom - and encouraged our students to challenge them. We gave them a paper by a respected author and admitted that we weren’t convinced by some of its arguments.

And we were delighted when they did challenge the notions we gave them. I could tell that they were excited by the idea that they might be discovering a world where there isn’t just one way to do things. A world in which they could make a difference.

This is not something you expect to feel as a graduate - I know this, I had been there myself. As a graduate you expect to spend a lot of time figuring out The Rules. You expect to know nothing in comparison to the eminent and respectable experts. You expect a long apprenticeship before you can even begin to contribute to your profession in a meaningful sense.
I don’t believe testing has to be that way, and I wanted to suggest as much to our graduates. By encouraging our graduates to discover testing for themselves, by treating their thoughts and ideas with encouragement and respect, and by challenging other - more canonical - notions of testing, I hope - and believe - we were able to do that.

The Heart of Darkness
Throughout the course, we were wary of deliberately indoctrinating the graduates to our way of thinking. In line with our overall approach, we wanted the graduates to make up their own minds about how testing might best be carried out.

As such, while our bias toward certain approaches will inevitably have shown through at times, we attempted to remain impartial in our delivery and specifically included a significant section of the course which allowed the graduates to experience and consider both a prescribed, verification focused and test-case driven approach to testing, as well as a more investigative and exploratory approach.

While it was difficult to include both approaches in good conscience, due to my strong preference for the latter testing mindset, it actually turned out to be a very interesting experience for me and my colleagues to witness ten unbiased (in testing terms), intelligent and critical minds engage in two very different approaches to testing.

The first thing I noted was that before we had given them any clues on how to test, they did so by exploring and investigating. While many of their early definitions for testing revolved around meeting business requirements, they immediately contradicted their own ideas when it came to an actual performance of testing.

We outlined requirements and they went beyond them, we asked if something worked and they told us ways in which it didn’t, and we watched as they discovered new information and used this to change and inform what they did next. None of them, before we introduced the notion of a test case, felt compelled to document the procedures they would follow before they began.

There was an energy about them too, when they were testing in this relatively unconstrained manner. We hadn’t told them anything about how testing might be done, and so they used their natural flair,
invention and instinct to drive what they did. The room was abuzz with discussion, excitement and occasional euphoric eruptions of discovery.

Then on the morning of the second day, we introduced them to test cases. In varying tasks they executed test scripts, and then wrote and executed their own and each other’s. Suddenly, it was like we were working with a different group of people.

The energetic, sometimes chaotic vibe from the day before was replaced with a nervous, suspenseful atmosphere. We went from classroom to exam room. Even their posture and focus changed - previously they were upright, interacting with their computers and their partners; now they were hunched, hushed, focused on their printed or hand-written test cases, their mice and monitors largely forgotten.

There are many theoretical arguments against the use of test cases as a default, best practice methodology for testing - they’re overly specific and reductive, cause inattentional blindness, infer dangerous and misleading levels of equivalence between variable test ideas, and are far too easy to count or gamify being just some - but here we had witnessed one which was fundamentally behavioural.

When, on the afternoon of that second day, we formally introduced the idea of exploratory testing to our graduates and again engaged them in practical testing challenges, their energy and enthusiasm returned almost immediately.

This was a relief - we wanted the graduates to enjoy themselves. We wanted their experiential learning to be challenging and difficult, but we wanted them to enjoy it. We wanted them to be engaged and relishing the challenge. But when they were using test cases, it didn’t feel that way. Instead of giving them a challenge, it felt like we were giving them a sentence.

**A Prevailing Wind**

It was this influence of a test case driven approach to testing that saw me discover, after a few months of my own career, that there was a different way to approach testing. I became bored and a little frustrated with simply writing tests to check requirements, and questioned whether there wasn’t *more* to the profession I had chosen. Luckily, there was.
My impatience with a test-case driven approach to lead me to discover the context-driven school of thought - where the mission of testing is less confirmatory, and more invested in identifying threats to the value of the project through whichever methods are most applicable - and the corresponding theoretical arguments against the prescribed, test case driven approach, as mentioned above.

This discovery was crucial in inspiring me to challenge the way that I and others were testing. It led me to implement a new approach to testing on the project I was working on and to collaborate with my aforementioned colleagues to challenge the way testing is done in our community.

However, once I had discovered an alternative to test cases in my own career, I had focused my thinking on the more theoretical side of the debate between confirming and exploring. I had also been fortunate in being able to change my own environment to reduce the boredom and frustration I’d felt. This experience with the graduates was therefore a welcome reminder of another important aspect of the debate.

Any approach to testing will be more effective when those performing it are engaged. This is true of almost anything in life - if we enjoy or are emotively or cognitively invested or challenged in an activity, we will usually give it more focused attention, concentration and effort than if we are doing it through a sense of duty or obligation.

In testing, the dangers from being disengaged in this way are great. James Bach describes testers as the headlights of a testing project, while Michael Bolton calls us its eyes and the ears - we are the senses by which projects discover information. But like a camera with a dirty lens, if we are not engaged with what we are doing we are going to be less effective in doing it - we’ll provide a fuzzier or less complete picture.

During the Graduate Programme we witnessed first hand the correlation between this loss of engagement and the adoption of a test case driven approach to testing, even in people who were young, fresh and enthusiastic. Through their behaviour and body language, but also in their specific feedback relating to the exercises - they admitted to boredom and frustration whilst using test cases, but their direct feedback described the exploratory sessions as fun and enjoyable.
It is important on two fronts then, that we are engaged in our testing. Not only is it essential in ensuring that we provide the best service possible for our clients, but it is critical in that most base of quests we all face in our daily lives: the pursuit of happiness. Approaching testing as an investigative and exploratory activity, rather than one of verification and confirmation, will likely make you a better tester, but it could also make you a more contented one.

Are we there yet?
This journey will, of course, go on.

The graduates have a great many adventures ahead of them, through which they will discover a great deal more about testing. They will likely also find that some of what they learned over the three days with us gets challenged or changed.

But for myself too, this experience was but the start of a journey. I not only have a great deal more to learn about testing, I also have a great deal more to learn about teaching testing, a pursuit I am particularly interested in, as it will prove crucial if we are to succeed in changing the way our craft is perceived and practiced.

Indeed, many of the ideas discussed in this article are things which I have learned only in the kind glow of hindsight and retrospect. Even those ambitions and techniques with which we approached the three days were but abstract or half-understood notions or instincts until I was able to observe them in practice.

If there is one lesson to take, then, from our whirlwind adventure around the world of testing, it is that the journey of discovery is never done - be you a novice, graduate tester or an old and wily one. The day you lose the instinct to explore and discover is the day that you stand still, left behind by a world in motion.

Adam Howard lives and works in Wellington, New Zealand.

After graduating university with an Honours degree in English Literature and Philosophy, he was as surprised as anyone to stumble into testing and discover an unexpected passion.

Three years on, Adam associates with the context-driven school of testing and has worked as a Senior Test Analyst and Test Lead on large scale, challenging and highly political projects.

He has had some successes in implementing lean and context-driven approaches to testing in his projects and is committed to challenging how testing is done in the wider testing community.

Adam writes regularly about testing on his blog and likes to stay involved in the testing world via Twitter (@adammhoward), and by attending meet-ups and peer conferences whenever possible.
JOIN THE TESTING TRAPEZE COMMUNITY ONLINE

Tell us what you think of our first edition, discuss and debate our articles, and discover a vibrant testing community:

www.facebook.com/testingtrapeze
www.twitter.com/testingtrapeze
www.testingcircus.com/category/testing-trapeze

WANT TO WRITE FOR TESTING TRAPEZE?

We welcome contributions from testers in Australia and New Zealand. If you would like to write for Testing Trapeze, please contact the editor: testingtrapeze@testingcircus.com.
EXPLORATORY TESTING SEEMS TO BE VERY popular these days, and though it fills my heart with joy, in recent years I have realised that people mean completely different things when they talk about it. Sometimes this difference is amusing. Sometimes, it's enriching and eye-opening. Sometimes - for example when your colleagues refuse to try something new, claiming that they are "already doing exploratory testing" - it's annoying.

I think it's a problem when people try to learn the practices of exploratory testing without understanding the ideas behind it. With an understanding of the ideas, you can choose or invent the way you do things, thinking about what would work the best for your current project and your team. If all you know is a set of tools, you are not exploring; you are asking for a script from people who did the actual exploration. This is not always bad, but in my opinion this is not exploratory testing. I don't claim to have the ultimate answer, of course, but I'd like to share my understanding of what exploratory testing is, and the reasons I love it so much.

Exploratory testing is not about using charts and session-based testing, it's not about working in an agile environment, and it's definitely not about some list of heuristics you absolutely must use. For me, exploratory testing is all about the idea of exploration. It's about study, research and investigation. It's about asking questions, experimenting, and gaining knowledge about the software under test. I like to think about it as if I were the Curiosity Rover or a crew member of the Starship Enterprise. I explore brand new worlds. I am aware that I will never know all I want to know, but I have the tools, the desire, and the attitude to learn.
As a tester, I start with some knowledge and assumptions about the software I’m about to test. I usually have the business requirements, a technology description, knowledge of the environment, general knowledge of IT, expectations for the software based on personal experience in dealing with the same types of tasks that software is supposed to accomplish, and so forth. I create a map based on my knowledge, which includes a test strategy and test ideas that I’m about to follow. I have many tools to help me on my journey, from specialised software to test analysis techniques. I need all these things to help me, but if I let them define what I do, my job will become mechanical. I wouldn’t need the power that is the human brain all that much if testing was about using some tools and following some scripts. When I’m doing exploratory testing, I have to keep asking questions and to adjust my assumptions. I’m amazed by how much you can gain from such a simple idea.

First of all, exploratory testing is highly efficient. By asking questions you gain a better understanding of the software. You make sure that you don’t use outdated documentation. You get to know people on a team and what they can do. You are providing fast feedback, which helps everyone on the team to do a better job by using more up-to-date information about the product than they would otherwise have.

Exploratory testing makes our job more fun, a point that you can't underestimate if you get bored as easily as I do. It also encourages you to use techniques from other fields - humans have been exploring the world since there were humans, and there is a large amount of historical experience and wisdom waiting to be applied to testing software. Think about the huge amount of literature on the scientific method, or about double blind testing in sociology, or about profiling criminals as they do in the FBI, or about planning a scouting operation on an enemy's territory - these are all examples of techniques from which you could learn how to do a better job as a tester.

If you agree with me that testing is about asking questions and gaining knowledge, you can use this understanding to deal with the project tasks that others try to enforce on you. I’ve often found that the product manager, or whoever is responsible for the application, presses the testing team into answering questions like "is it ready to go into production?" and "when will it be ready?". The problem is that the
testing team doesn’t have the power to influence the situation much. They can’t decide what is acceptable to have in production, they can’t assign additional time for testing or for development teams, and they can’t rush developers, designers, translators and others into doing their jobs. And if you have no power, you can’t take the responsibility.

All these ideas form a practical approach that I’ve been applying for years, even before I had the terminology to talk about it. Here are the main points of exploratory testing as I see it, in a series of statements.

1. The mission of software testing isn’t to provide quality, it’s to gather and provide helpful information to those who make decisions about the product and its perceived quality.

2. Exploratory testing is an approach that doesn’t depend on the chosen development model. It can be applied to any situation, even outside of software development.

3. It’s a bad idea to commit to any exact long term evaluations on how much time you need to ‘do the testing on a project’, because testing often includes walking in a dark room that keeps changing. Make sure your superiors know what can influence the testing process and change time evaluations.

4. Exploratory testing consists of many iterations of two very different steps: discover and investigate. Each step has its own challenges and goals. In discovery you are concentrated on finding issues, and in investigation your goal is to gain the information you need to deal with that issue.

5. Use test ideas to guide testing rather than test cases to define it. It is important to ask questions and to look at problems from different angles, to use new tests instead of old ones.

6. Use automation to help with mundane tasks and free your precious time for smart tasks - not to replace all manual testing there is on a project.

7. Give feedback as soon as it can answer questions relevant to the project. Does this feature do what it is supposed to do? Is this bug fixed? Give clear and specific feedback, and make sure it isn’t personal or insulting.
8. Prioritise your work: decide which features to test first, how much time to spend on a problem, which risks to mitigate and which tasks to do first, and so on. Risk management is a nice basic tool to help with this.

9. Know your tools: heuristics, practices, diagram notations, software that can help on the job, and so on.

10. Know the software you are testing from both the business and technological points of view; what problems is it supposed to solve, who are the stakeholders, and how does it work?

11. Don’t spend time on documentation no one will read or use. Keep documentation that is in use up-to-date. Don't keep your experience and knowledge of the project in your head.

Last thing I’d like to touch in this article is the common notion that exploratory testing only works for experienced testers. I guess we can blame ISTQB for that since that is exactly how they classify exploratory testing. My experience as a tester and a test lead shows that this is not true. A tester who is completely new to a product and/or software testing itself can learn exploratory testing and start using it immediately. I think the only thing that is absolutely required is passion for asking questions, everything else will come with the answers to those questions. I have trained new testers in an exploratory approach instead of the classical "do step-by-step test cases until you learn them by heart" approach. I think they are now amazing testers.

Once again, this is just my view on exploratory testing. There are many other testers who write on this subject. I would like to acknowledge the writers of blogs that I have read and learned from over the last eight years, but there are too many names to list. I’d like to specifically thank James Bach and Michael Bolton for their brilliant Rapid Software Testing course, which gave me ideas to think about, confidence in my approach, and terminology to use to vocalize my thoughts on testing.

Viktoriia has been testing software for more than nine years.

She tried different roles throughout that time, going from manual tester to automation engineer, to a test designer, to a test lead, to a product manager, and back to a hands-on testing role. For the last three years she spent the most time testing mobile applications in Saint Petersburg office of Yandex, and now she is working as a performance engineer in Orion Health in Auckland.

Viktoriia is a big fan of sci-fi, science, board games, exploratory testing and finding efficient and fun ways to do stuff. Originally from Russia, she has just relocated to New Zealand last August, and she is excited about getting to know the testing community here. You can say hi to her via Twitter (@miss_hali).
NEXT TIME IN TESTING TRAPEZE:

ANNE-MARIE CHARRETT
SYDNEY, AUSTRALIA

MIKE TALKS
WELLINGTON, NEW ZEALAND

DEAN MCKENZIE
BRISBANE, AUSTRALIA

MICHAEL TRENGROVE
CHRISTCHURCH, NEW ZEALAND

ALESSANDRA MOREIRA
FLORIDA, USA

AVAILABLE
APRIL 15TH 2014
TEST CASES ARE NOT TESTING:
TOWARDS A CULTURE OF TEST PERFORMANCE

JAMES BACH
WASHINGTON, USA
AARON HODDER
WELLINGTON, NEW ZEALAND

“We are seeking a Graduate Test Analyst to write and execute test cases to ensure quality is delivered to an extensive client base”

“Responsibilities: Assist with requirements gathering and develop test cases that satisfy the requirements”

“To be successful in this role, the following is required:...To have written manual test cases and involved in test execution...Proven experience in writing Test Conditions & Test Scripts”

“Responsibilities:...Create comprehensive test cases relevant to test conditions...Execute test cases and report results”

From job advertisements listed on seek.co.nz on 24th January, 2014.

OUR INDUSTRY HAS A TROUBLING OBSESSION WITH TEST CASES. WRITING TEST CASES IS mentioned in job descriptions as if it were the main occupation of testers. Testers who approach us for advice too often use phrases like “my test cases” to mean “my work as a tester.”

This test case focus has an innocent basis, and is for the most part well-intentioned. But it has become toxic to the field and we, the authors, believe it’s one continuing reason why testing commands so little respect. Test cases are holding us back.

It’s time for an intervention. In this article, we will critically examine the notion of a test case, and the culture that so often surrounds it. We will show why testing cannot be encoded in test cases, then suggest an alternative vision based on testing as human performance, rather than on artifacts.
What is a test case?
Definitions vary from place to place. One common idea of a test case is that it is a set of instructions and/or data for testing some part of a product in some way. Testers will speak of test cases that are written, or about to be written. In some projects, a test case will always have a unique procedure associated with it. In other situations, test cases may share procedures and be distinguished by unique data.

There is little difference between Agile and non-Agile projects when it comes to test cases. Test cases in Behavior-Driven Development are often specified with a formal structure for execution by tools such as Cucumber. Agile projects tend to be more automation-oriented and more focused on using test cases to define “done.”

Here are two simple, contrasting examples:

First, consider a table of test cases characterised by differing conditions. The procedure associated with these cases is implied or documented elsewhere. Thus, test cases may be spoken of as variations on one test idea. Often the contents of each cell are used to report the status of the product with respect to that case. Note that it’s not obvious how to delineate or count the cases in this example. “Cookies Accepted,” being a distinct idea, may fairly be called a test case, or perhaps each cell in the table is a test case.

<table>
<thead>
<tr>
<th></th>
<th>Chrome</th>
<th>Firefox</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cookies Accepted</td>
<td>Pass</td>
<td>Pass</td>
</tr>
<tr>
<td>Cookies Not Accepted</td>
<td>Pass</td>
<td>Fail</td>
</tr>
</tbody>
</table>

The second example is also commonly called a test case. It is a step-by-step procedure:

<table>
<thead>
<tr>
<th>Step</th>
<th>Expected Result</th>
<th>Actual Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Start Chrome browser</td>
<td>Browser starts</td>
<td>Pass</td>
</tr>
<tr>
<td>2. Set to accept cookies</td>
<td>Browser accepts cookies</td>
<td>Pass</td>
</tr>
<tr>
<td>3. Attempt log in</td>
<td>User home screen displays</td>
<td>Pass</td>
</tr>
</tbody>
</table>

As above, each step might be called a test case because each step has a verification operation associated with it. There is no objective, universal way of accounting for test cases, and hybrids of data-like and procedure-like cases are often found, wherein a procedure-like test
case includes variables that take their values from data-like test cases stored in tables.

These are not the only kinds of things called test cases. A tester can make a loosely structured list of ideas such as “load a corrupted file” and call them test cases. Considering the variety of things called test cases around the industry, a definition that covers all of them would have to be quite general. However, our concern in this article is mostly with detailed, procedural, documented test cases, and the attitudes surrounding that kind of test case.

Note: Often test cases are poorly designed. James was once ordered to create test cases by adding the words “verify that...” in front of the literal text of each requirement. But that silliness is not our complaint in this article. The issues we are raising hold even if you assume that each test case is well-designed. Our claim is that even good test cases cannot comprise or represent good testing.

The Innocent Foundation

Programmers write code. This is, of course, a simplification of what programmers do: modeling, designing, problem-solving, inventing data structures, choosing algorithms. Programming may involve removing or replacing code, or exercising the wisdom of knowing what code not to write. Even so, programmers write programs. Thus, the bulk of their work seems tangible. The parallel with testing is obvious: if programmers write explicit source code that manifests working software, perhaps testers write explicit test cases that manifest testing.

It is seductive to think of written test cases as the “code” of testing. We may covet the sense of accomplishment that comes from producing a tangible asset. We may delight in the simplicity of direct correspondence between test cases and written code. There certainly are situations where thinking in terms of detailed, explicitly specified test cases is appropriate. For instance, when we need to cover a function that can be described cleanly and systematically in terms of a few interacting variables, it can be sensible to model that space formally and then formally specify which points in the space to test. Or perhaps if we want to carry out an intricate and specific test, or a test that requires several testers to coordinate their actions, or even a set of simple fact checks that must be performed periodically - in any of these cases it can help to encode them down step-by-step. And of
course, if the operation will be performed by a machine, it must be encoded.

But this becomes fertile ground for a vicious cycle: in some specific situations, managers may ask to see test cases for some valid engineering reason, and testers may deliver those cases; but soon providing test cases becomes a habit, and then a tradition, and then someone starts using the term “best practice” as if habitual behavior had won some sort of world championship. The goal of good testing becomes displaced by a blind mandate.

Test cases are not evil. Neither are french fries nor chocolate candy. The problem is the obsession that shoves aside the true business of testing. Chocolate-covered french fries have become the staple diet of most of our industry. In too many organizations, testing is fat and slow. We would like to break the obsession, and return test cases to their rightful place among the tools of our craft and not above those tools. It’s time to remind ourselves what has always been true: that test cases neither define nor comprise testing itself. Though test cases are an occasionally useful means of supporting testing, the practice of testing does not require test cases.

Test Case Culture and the Factory School

Obsession with test cases is not just a habit, though. It is embedded in a culture.

Aaron once ran a small experiment at a software testing course he was attending. He asked fellow attendees whether they write test cases before they start testing. He expected the answers to range from “Yes, of course” to “No” with a healthy dose of “What do you mean by test cases?” thrown in. To his shock, the majority of respondents just looked at him quizzically as if he had just asked them whether they wear clothing to work, or whether they hold their breath while swimming underwater. Test case writing as a central practice appears to go unquestioned in a lot of organisations.

A test case culture is not one that merely encourages using test cases as a tool to support testing. In a test case culture, the test cases are equated to testing; testing is viewed as a mechanistic, clerical task of executing test cases (analogous to the mechanistic way that a compiler
turns source code into object code) for the purpose of checking specific and easily observed facts about the product.

A test in a test case culture is a concrete noun, an artifact you can point to and say “that is a test.” Once reified in that way, it is a natural step to treat “tests” as a commodity, like so many sacks of rice. We often see tests counted and testing progress communicated solely in terms of such numbers. Bugs are subject to the same reification; thirteen bugs is one worse than twelve bugs, right? In this culture, bugs and test cases are linked - after all, test cases find the bugs. If a bug is somehow found without a test case (a situation test case culture views with suspicion), we might expect some manager to ask for a new test case to be created that exhibits that bug.

In a test case culture, the tester is merely the medium by which test cases do their work. Consequently, while writing test cases may be considered a skilled task, executing them is seen as a task fit for novices (or better yet, robots).

This way of thinking is attractive because it seems to allow testing to be managed with an unambiguous accounting system that makes testers into fungible resources; as a sort of factory of testing. Hence we often call this the Factory School of testing thought.

A common attitude about process in factories is that there is one right way; and that this right way should be defined and followed. But how that process is discovered is completely outside the scope of factory thinking. In test case culture, this leads to a cartoonishly simplistic understanding of test design. A common phrase in that culture is that we should “derive test cases from requirements” as if the proper test will be immediately obvious to anyone who can read. In test case culture, there is little talk of learning or interpreting. Exploration and tinkering, which characterize so much of the daily experience of engineering and business, are usually invisible to the factory process, and when noticed are considered either a luxury or a lapse of discipline.

A common attitude about people in factories is distrust. People are unreliable at following the one right way. People are, at best, a transitional technology: they are tolerated until the right kind of drones can be built. But even in the most successful factories you will notice...
that the managers don’t consider replacing themselves. At some level, they acknowledge that human attention and action is required. Indeed, here the authors find common ground with the Factory-schoolers that there is such a thing as necessary humanity - except we contend that humanity is necessary at the level where we perform the test.

**Factory Theory Meets Practice**

What happens when we try to manage a complex cognitive activity such as software testing by reifying the activity down to a superficial representation such as test cases?

Recently Aaron observed a class that aimed to teach the limitations of test cases. The students were given eight identical test cases and were instructed to execute them. In Aaron’s opinion, these were reasonable examples of relatively good, unambiguous test cases. At the end of the exercise, students were asked for the number of test cases that had passed, the number that failed, the number unexecuted, and the number of bugs found.

One of the purported benefits of a test-case driven approach is that consistency and repeatability are ensured. Aaron expected to see some variation in the results to demonstrate that these claimed benefits are unfounded, but the results were even more striking than anticipated.

Some groups reported no bugs at all, while one group found five. While a few groups reported two bugs, upon further elicitation it was discovered that they weren’t necessarily the same two bugs.

<table>
<thead>
<tr>
<th></th>
<th>Pass</th>
<th>Fail</th>
<th>Unexecuted</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>0</td>
<td>7</td>
<td>1</td>
<td>8</td>
</tr>
</tbody>
</table>
This exercise demonstrated beyond even his initial expectations the danger of assuming that test cases can be a reliable medium of conveying testing ideas and reporting on quality-related information. Each group, even though they had the same test cases to work from and were instructed identically, brought their own judgements and heuristics to bear resulting in eight different testing performances.

To report on testing by reducing the individual performance down to a numerical report of test cases passed vs. failed is at best, of marginal value, and at worst, potentially misleading. Yet, this is the primary way testing is managed in a lot of organisations.

This case is not a carefully controlled scientific study, but at the very least this experience undermines the glib assurances by many testing consultants and authors, going back to Program Test Methods, the first book on testing, written by William Hetzel in 1973, that written test cases provide a strong and stable basis for testing.

**Testing is Not a Factory. Testing is a Performance.**

Testing is an event; an activity; a performance. If we work backward from the moment that a tester successfully reports an important problem, we find it results from many overlapping and supporting thoughts and judgments and experiences. There are innumerable ways that this process can play out, all of which require human enactment.

Testing is the evaluation of a product by learning about it through experiment; by seeing it in action. The reason we test is to analyse product risk: the danger that the product will cause trouble for its users or otherwise fail in some way to fulfill its purpose. In other words, we look for anything about the product that might significantly impair its value. We are looking primarily for “bugs.” We want to find every important bug, although there will be no way to know for sure that we have succeeded.
Testing as a performance depends on the credibility of the performer, which is improved or damaged in every interaction with the team. But excellent testing is a complex performance that is difficult to teach, supervise, or evaluate. While novice testers may find some bugs by romping around like kittens, management needs confidence that there has been a diligent search for important problems. This confidence must come, in large part, from the personal credibility and observable behavior of the responsible tester who vigorously questions the product.

Can an algorithm exist that will guarantee we find all the important bugs? No. This is a matter of basic computing theory (see the Halting Problem) and the fact that bugs are socially constructed by users rather than being something to do with the essence of the product. Bugs are not “in” the product. Bugs are about the relationship between the product and the people who desire something from it. It’s possible for a bug to be created or resolved just by changing the stakeholder. And apart from every other problem, a total bug identification algorithm would require a complete, unambiguous, and up-to-date specification that is accepted by all stakeholders... and when was the last time you saw one of those?

When we test we are exploring the relationship between the product and values about the product. To do so, we must make many social judgments, including judging the importance of testing a specific situation, and judging the meaning and importance of potential problems.

This seems hard to accept for many casual observers, though, because many kinds of bugs seem so obvious and uncontroversial. Indeed it is possible to create algorithms to detect specific kinds of problems that possess identifiable and predictable characteristics. Many such checks are already built into compilers and application frameworks.

However, even if all imaginable checks are performed, there is no theory, nor metric, nor tool, that can tell us how many important bugs remain. We must test - experiment in an exploratory way - in order to have a chance of finding them. No one can know in advance where the unanticipated bugs will be and therefore what scripts to write. The
performance of testing unfolds forward in time, like a swarm of ants foraging for food. This is a constantly shifting picture.

Although a tester can and should prepare to perform, testing proceeds in ways that cannot be predicted simply because of two things: there is always more testing to do than we can afford to do, and we don’t know where the bugs are until we discover each one. This unpredictability requires each tester to be prepared to live in and react to the moment, regardless of any specific plan. That’s why, just as flying an airplane, doing surgery, or playing football are rich, complicated performances, testing is too.

**Why Testing Cannot be Literally Encoded**

Distilling test ideas from tacit mental models; explicitly and precisely describing test cases; writing them down: let’s call that “encoding.” Encoding means the expression of ideas, explicitly, in the form of some sort of code (such as written words or software). Test case culture insists on encoding testing to such a degree that only trivial aspects of the process should remain undeclared. It insists that encoding is practical and desirable - moreover, that it is necessary; that it is irresponsible not to encode testing.

But this is not so, because testing cannot be encoded.

There is no support in scientific or engineering literature for the idea that testing can be encoded. Despite years of searching, the authors are aware of no studies that have ever shown that testing should be written down, or even that it can be written down. In fact, the opposite is more the case. See the *Sciences of the Artificial*, by Nobel laureate Herbert Simon for a deep treatment of this topic. Simon shows that perfect rationality is unavailable to us in any but the simplest situations, and explores the nature of design processes as “bounded rationality” requiring heuristic solutions. Or perhaps *Introduction to General Systems Thinking*, by Gerald Weinberg, who shows that observing and describing systems requires us to simplify them, and that there is no algorithm for knowing how to do that without losing something that might be important. Or look at *The Social Life of Information*, by Paul Duguid and John Seely Brown, who tell how copy machine repairmen learned their craft not by reading the official documentation but rather by socializing with each other in free-form ways. Or *Things That Make Us Smart*, by Don Norman, who shows how adding a cognitive artifact
(such as a test case document) to a process changes the process in potentially unpredictable ways. If you even glance through any of these works, you will see a rejection of mechanistic, reductionistic, algorithmic ways of conceiving and controlling complex systems, including social systems.

In performance terms we consider a test, in its noun form, to be the act of configuring, operating, observing, evaluating some part of a product in the service of a test project. So, what testers call a test is a process embedded in a larger process called “testing” that includes reading specifications, attending meetings, acquiring equipment, etc. The scope of a test is elastic. It may involve the speculative exploration of a product, or something as simple as checking the result of a function call. There is no objective method by which we can draw sharp lines between individual tests: it is purely a matter of convenience and context how you choose to delineate them.

Testing has many levels, all of which contribute to the success of the testing enterprise, and very little of which can be encoded:

1. **A person is born.** Yes, it’s important to start here. Each of us has a specific genetic, environmental, and cultural foundation that means we approach testing with a certain mix of talents and a certain temperament. The fact that James is mathematically inclined leads him to be biased in favor of analytical modeling the things he tests. Other testers may approach the work in a more intuitive or social way. There is no such thing as purely objective and unbiased testing. Two test designers, unlike, say, two car engines, cannot be analyzed and compared in terms of any universal model of testing performance. *Testing talent and temperament cannot be encoded.*

2. **A person learns to test.** Some skills useful in testing are ubiquitous among adults. Others come with general technical or scientific education. Some are technology-specific, and some are specific to testing itself. Learning to test begins in childhood as we play and interact with our world. Testing skill can be acquired in a variety of ways, but deep systematic training in testing is difficult to obtain.

There is no governing body for the testing field. Therefore, there is no generally accepted Body of Knowledge, or taxonomy of required skills. Commercial certification programs are controversial, but even
if one accepts their view of testing, they do not even attempt to assess practical skill. It is not unusual for people with no significant experience in testing to become a certified tester. This is not possible in reputable fields such as medicine or air transport.

The result is that testers vary quite a bit in their practices and their grasp of different aspects of testing. Each tester’s education is local, conditioned by the idiosyncrasies of specific technologies, companies, and projects. And on top of all that, much of testing skill is comprised of inherently tacit skills such as questioning, collaboration, and systems analysis, which cannot be made explicit and mechanical.

No one even attempts to encode the fine details of their own testing skills.

3. **A tester joins a project.** Joining a project is a complex social event. As part of that process, we learn whom we serve. We come to understand the scope of our mission as testers. We commit to the project. This context conditions everything we do as testers. While a mission statement and elements of context can be written down, there are innumerable ways that one might interpret those things and act accordingly. There is no calculus for determining how all that influences testing.

A tester’s sense of and response to context can be sketched, perhaps, but not fully encoded.

4. **A tester learns the product.** Each of us must construct a mental model of the product, its context, and its uses. A more familiar way of saying that is we have to learn all about it, and the result of that learning is a mental structure from which we can design tests. This model itself cannot be encoded in any explicit form (it’s neurons, baby). But we can, if we choose, produce some formal and explicit projection of our mental model.

Therefore while some of our learning can be encoded, most of it will never be, for at least two reasons:

   a. We have no algorithm or mechanism for doing a “brain dump” that accurately reflects the state of our knowledge about anything.
That means we can never rule out the possibility that there is a fact we know about and yet have not put into our model.

b. Even if there were such a mechanism, the totality of what we learn is overwhelming. For instance, to list our true expectations for the behavior of a browser that displays a simple webpage, such as Google’s home page, would require prohibitive time and space.

Oh, and that doesn’t include an even bigger dynamic: the process of learning the product is also testing. The product is operated, observed, and evaluated during the learning process. Therefore, even to the degree that some aspects of testing can be formalized, that can only occur after a substantial amount of un-encodeable learning and exploring work has already been done.

Therefore, the bulk of our product learning, and testing while learning, cannot be encoded.

5. A tester enacts the testing according to some idea. This is the act of experimenting on the product, apart from all the processes of preparation that support or inform it.

It is traditional to divide testing into test design, test execution, and result evaluation. Test design may be further partitioned, perhaps into coverage modeling, data modeling, procedure design, oracle design, and tooling. Perhaps this tradition has not served us well in one respect: it seems to imply that there are clear divisions between these activities, and that they are independent of each other. This is not the case. Although when training testers it often helps to focus on each of these in isolation, the practice of testing brings them together in an evolving, exploratory process.

This interplay cannot be encoded. The most we can do is write extensive notes about our thinking in every session of testing, but writing it down, beyond a certain point, interferes with testing. And even perfect notes about our thinking would not be an encoding of the process of that thinking - in other words we can’t write a program, while we are working, that duplicates the workings of our minds.

Picture the process of testing: You look; ponder; try something. You see what happens and ponder that. You have a question, then
conceive of an observation that might answer the question. And so on. Skilled testers perform hundreds of what might seem like discrete tests in a session. Few of them need to be repeated. Most tests performed are informed by the results of the previous test. The value of a test may not be known until it has been performed, or possibly much later.

Testing seems encodeable because we can crystallize - from out of this thinking-learning-trying soup - acts of configuring, operating, observing and evaluating the product. These moments are embedded in our evolving concept of risk and of the status of the product. But to an outsider, who is not privy to the workings of a tester’s mind, they may seem to stand alone. James Bach and Michael Bolton, in their Rapid Testing Methodology, call such acts “checks” if they can be performed, in principle, by a machine (http://www.satisfice.com/blog/archives/856). It’s useful to talk about checking because it is a task, embedded within testing, that might be accelerated or substantially supported with automation.

But we must always remember that checking does not represent or comprise testing itself, just as a hammer does not comprise carpentry. We can encode a check, by definition. We cannot encode the process of conceiving, designing, implementing, re-evaluating, or judging the meaning of the results of performing a check.

Some would say we can encode testing simply by recording keystrokes or videoing the test process. Those recordings can be helpful, but they are mere echoes and hints of the testing thought process. They don’t encode the richness of the bug seeking and finding intelligence and experience. At no time, when replaying keystrokes, will your test tool stop and say “wait a minute, I think I’m looking at the wrong thing.”

Expectations also cannot be encoded. In James’ classes, he demonstrates that by asking students to list their expectations for the output of a simple and well-understood feature of an everyday product. Then James proceeds to list dozens of expectations that each student agrees with - but did not think of listing. If testers who try very hard to list expectations can’t do it completely even for the simple functions, it is outrageous to think that testing can be fully encoded.
Therefore, with the exception of certain acts of fact checking, enacted tests cannot be encoded.

6. A tester reports, explains, defends, and amends testing. Testing doesn’t end with the output that the system-under-test produces. The results must be made relevant to the project. This process of reporting bugs and status and concerns happens throughout the test performance process, and it influences that process. It is probably not sensible to separate this process from the operational performance of testing.

A test report may be partially encoded, but there are innumerable judgments to be made about what to say and what to keep silent. Reporting involves responding to questions, too. There is no way to encode an algorithm for that. The act of reporting may spur testers to redo or add to the testing performance, and that also cannot be encoded.

Yes, bits of testing can be encoded. We can use encoding to create useful anchor points for the test process. We might use test cases or other kinds of lists, diagrams, or references to formalize parts of testing. These should be considered tools that support testing, not testing itself.

We contend that factories don’t apply to testing. While industrial factories are productive (say what you want about how the iPhone is manufactured - you can’t deny that it IS manufactured) testing factories do not work. Testing is not manufactured. Testing factories are a big lie. Test case culture is a ceremonial approach to testing. It is, quite simply, fake testing.

But How Can Fake Testing Seem to Work?
Answer #1: testers may be secretly not faking it. Brian Osman dubs this practice “stealth testing.” This is skilled testing, done for the good of the project, kept hidden due to a management culture that demands performance while at the same time mandating processes that undermine performance. Stealth testing, while well intentioned, helps to perpetuate the test case myth. This is a double-edged sword. If stealth testing finds important problems, and finds them quickly, the tester doesn’t get the credit; the approach they actually used doesn’t get the credit. The publicly avowed process gets the credit.

Answer #2: the product may be good enough even with poor testing. Quality comes mainly from
developers who do a good job. A wasteful test process might amount to little more than a sanity check, and yet the product simply has no important, deep bugs to be found. And face it, many products are pretty bad, and yet still put out there to torment users. The market for software is not an efficient one with respect to quality.

Answer #3: it is easy to shift the blame for it not working. Ironically, the inefficiency and ineffectiveness of test factories can be used as an excuse to invest more in them. Once a gullible business has been convinced that testing must be structured in test cases, then any problem that escapes testing seems due to not having enough test cases. If you believe in test factories, any problems are either down to the factory not being big enough, or bad people who are sabotaging it. Yet, behind the expensive high walls of test case documentation and the publicly avowed processes that go with them, broken practices of testing can easily hide, and there is little incentive to improve.

Toward a Performance Culture
A performance culture for testing is one that embraces testing as a performance, of course. But it also provides the supportive business infrastructure to make it work. Consider how different this is from the factory model:

- **Testing Concept:** Testing is an activity performed by skilled people. The purpose of testing is to discover important information about the status of the product, so that our clients can make informed decisions about it.

- **Recruitment:** Hire people as testers who demonstrate curiosity, enjoy learning about technology, and are not afraid of confusion or complexity.

- **Diversity:** Foster diversity among testers, in terms of talents, temperaments, and any other potentially relevant factor, in order to maximize testing performance in test teams.

- **Training:** Systematically train testers, both offline and on the job, with ongoing coaching and mentoring.

- **Peer-to-peer learning:** Use peer conferences and informal meetups to build collegial networks and experiment with methods and tools. Occasionally test in group events (e.g. “bug parties”) to foster common understanding about test practices.
• **Openness:** Foster the ability to narrate, explain, and defend testing performances. Create a culture of normalcy about working together and sharing work.

• **Transferring work:** One tester may take over from another with the help of concise documentation, discussion and demonstration, or simply by starting from scratch. Among skilled testers, this is rarely the problem that non-testers fear it will be.

• **Personal Excellence:** Testing depends upon testers who have pride and integrity in their work, and who strive to learn their craft. Part of the reason performance culture is not more accepted in the industry is the lack of trust by management that testers will perform.

• **Team Integration:** Foster a mutually supportive attitude between testers and development. As trust develops, everyone’s performance becomes more fluid and collaborative.

• **Preparation:** Detailed and meticulous planning is rarely cost effective in a high innovation environment such as software development, but that doesn’t mean we can’t benefit from good preparation. Learning about tools and technologies and developing test ideas in concise form is part of performing at our best.

• **Responsiveness:** We recognize that time is of the essence. We look at the product as soon as it is available, and if someone taps us on the shoulder and asks "How did the testing go?" we strive to answer with useful information, confidently and immediately.

• **Cyclic, Exploratory Process:** Performing feeds on itself. When we test, we are also uncovering better test ideas as we go.

• **Agility:** In performance culture, agility is easier, because we aren't traveling with all that baggage of documentation. That means we can respond more rapidly and productively to changing context.

• **Metrics:** Metrics may be used to provoke inquiry, but do not use them as the basis of decision rules to control a social system such as testing. Any metric put in place by management to control people will be used by people to control management.
• **Documentation:** Prefer concise documentation, such as lists and mind-maps, that are less expensive to produce and maintain. Develop a discipline of personal note-taking.

• **Management:** Test leads must supervise junior testers or have them work with senior testers until they are ready to take full responsibility for their own performance. First level management must be involved in testing on a regular basis. Give autonomy to qualified testers to choose their style of work. Celebrate successes, but also celebrate honest, hard-won failures.

• **Process control:** Focus on heuristic rather than algorithmic process controls. Focus on discussion rather than numbers. Focus on trusting people who have earned credibility rather than on inanimate controls and surveillances. If more formal controls are needed, consider using an activity-based approach such as session-based or thread-based test management.

• **Regression Testing:** We may use automated checking tools to help detect obvious problems at each build. These tools are supervised by testers who take responsibility for them. In addition to any checks, however, regression risk may require the tester to enact new tests, or refresh performance of previous tests.

• **Tools:** Use tools under the direction of testers in ways that augment any aspect of tester performance. Do not equate checking tools with human cognition.

• **Stopping:** Testing is finished when the clients of testing feel that every important question about the status of the product has been answered. This feeling is arrived at by discussion of the testing and test results throughout the project.

The Context-Driven testing (CDT) movement, has, for years, been promoting a humanist, performance-oriented vision of testing. There are now two international organizations and several conferences devoted to CDT. Although CDT is not against any practice, it is against methodological chauvinism. The practices we use should be the practices that work well and fit the context. It is through ongoing study and skeptical self-examination that we free ourselves from bad habits and inappropriate practices.

After more than 40 years of trying, the factory approach to testing has not solved the world’s testing problems. Enough is enough. Abandon the swamped, lumbering barge of test case culture. Re-discover testing as an intellectual pursuit. The complexities and risks of our world demand that we do this.

*The authors thank Michael Bolton and the Testing Trapeze review team for their invaluable review and comments.*
Thirty-one years ago James was a high school dropout working as video game programmer, writing in Assembler for the Commodore 64. A few years later his “big break” came when he was hired to manage a test team at Apple Computer. He’s been hooked on testing ever since.

He spent nearly a decade as a test manager in Silicon Valley during the 90’s, and became well known for rethinking software testing to give it a grounding in social science and general systems thinking. The focus of his work was much influenced Cem Kaner and Jerry Weinberg.

James is a founder of the Context-Driven school of testing. He created and teaches the Rapid Software Testing methodology, and has written two books: Lessons Learned in Software Testing (with Cem Kaner and Bret Pettichord) and a book about succeeding without going to school called Secrets of a Buccaneer-Scholar.

Aaron Hodder is a context-driven tester from Wellington, New Zealand, with a true passion for the software testing profession. He is an advocate for structured exploratory testing techniques such as session-based and thread-based test management, and visual methods of test reporting.

Active in the software testing community, he co-founded WeTest work shops in Wellington with Katrina Clokie, and regularly tweets (@AWGHodder) and blogs about testing at TesterKiwi and HelloTestWorld.

Aaron has spoken at STANZ and CAST conferences about using mind-mapping software as a testing planning and communication tool, and has attended the invite-only Kiwi Workshop on Software Testing peer conference all three years it has been running.
THE TRAPEZE ARTISTS:

EDITOR
KATRINA CLOKIE

DESIGN & LAYOUT
ADAM HOWARD

REVIEW PANEL
AARON HODDER
ADAM HOWARD
BRIAN OSMAN
DAVID GREENLEES
OLIVER ERLEWEIN

COPY EDITOR
RICHARD CLOKIE

Testing Trapeze accepts no liability for the content of this publication, or for the consequences of any actions taken on the basis of the information provided.

No part of this magazine may be reproduced in whole or in part without the express written permission of the editor.

Image Credits: cover, contents, pg.2, pg.9, pg.15, pg.18, pg.24, pg.29, pg.48