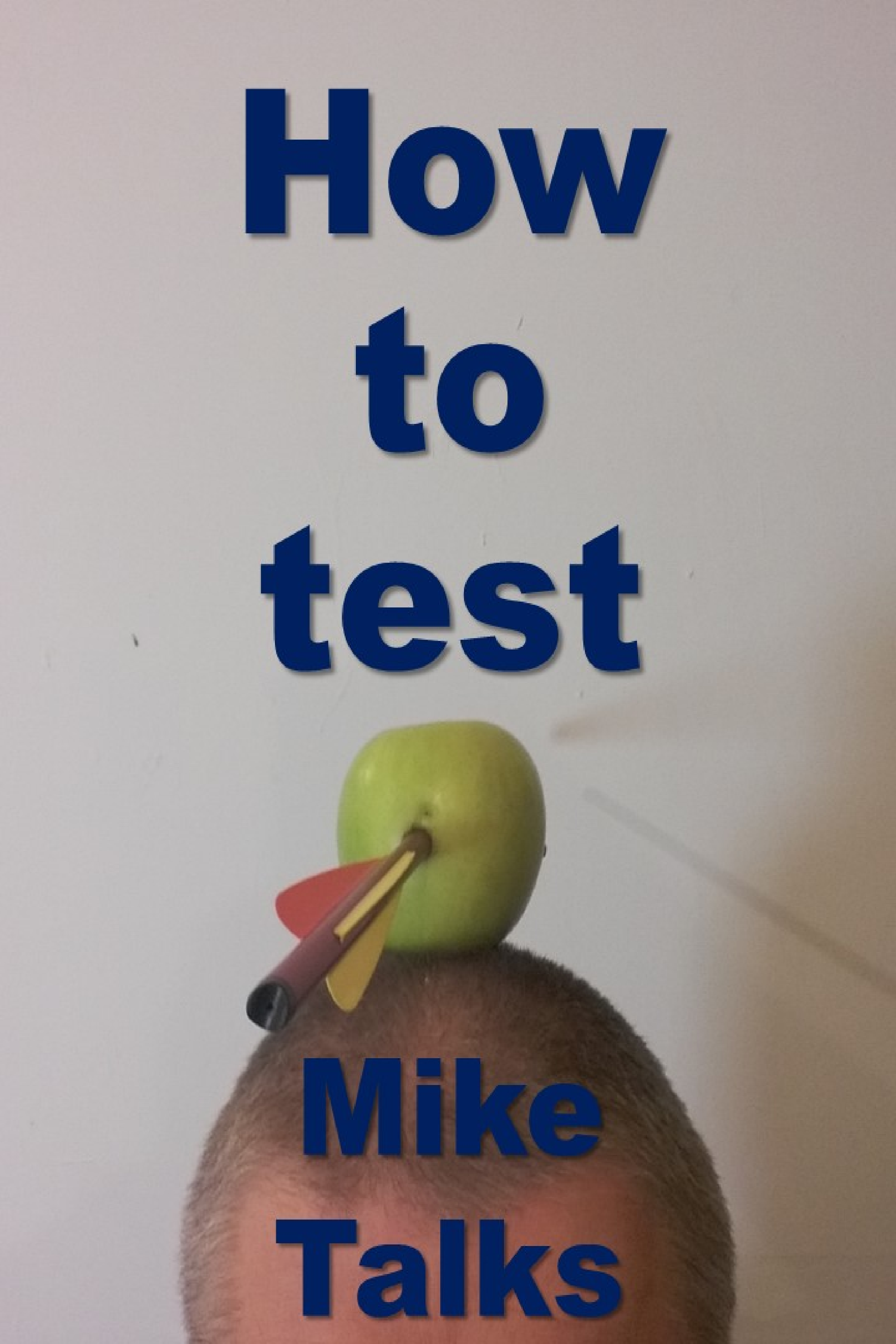


How to test

Mike Talks

How To Test

Mike Talks

This book is for sale at http://leanpub.com/howtotest

This version was published on 2016-04-14

Contents

Dedication

To nurture is to love. In memory of my beloved grandmother, Elsie Taylor, who always saw the potential in me.

Introduction

A few years ago, I was exchanging emails with another test manager named Bernice Ruhland - and we were having a similar problem. How do you help to develop someone who is new to testing?

The problem is this - there are a lot of books out there, but they can be quite long, and they represent a range of opinions. Oh yes, there is a spectrum of feeling about “how to test”, let me start by saying that. There are also a lot of training courses out there, some claim they can certify you as a professional tester, for a hefty price. However some of those courses are not the automatic entry into the profession that they claim to be.

We wanted to make a quick, easy-to-work through guide for people new to testing. Our target would be people who were considering a career in testing. But it would also be of use of people within IT who wanted to know more about the core skills of testing.

The book also needed to be free. Both Bernice and I have received a lot of help from other testers around the world on a spectrum of subjects, and we wanted to *"pay it forward"*. Neither does the book end with any recommendation for any book or project we have a vested interest in.

We had great plans, but never quite the time to write it. Thanks to an invitation by The Summer Of Tech in Wellington to teach a workshop on testing to a group of technology students, I finally got the kick-in-the-ass I needed to sit down and write, and this book you're reading is the end-product.

I hope it gives you the material you need to think, experiment and try out the core skills and thinking of the testing profession. If you read this and want to know more, there are some suggestions at the end for next steps.

Some people see testing as a potential easy in-road to the world of IT and potentially programming. But to myself and others, software testing is it's own unique and challenging career path which is full of it's own reward.

Just remember - everyone benefits from a kick-in-the-ass at some point

Mike Talks

September 2015

How to use this book

Yes, I'm quite certain you know how to use a book, but this one is a little different. It's been designed with two kinds of end user in mind.

To the student tester

So you're just starting out, or thinking of starting out on a career as a tester. Welcome!

This book is designed to teach you some of the nuts and bolts of testing - the day-to-day tasks of testing. As such there is some material which I have deliberately missed out - talking in detail about software development lifecycles such as waterfall or agile for instance. Iâ€™m really trying to focus on the skills and knowledge needed to get you up and contributing as a manual tester on a team.

Some of the sections include proposed activities for you to explore, trying out different social media registration, login and account management. I really encourage you to have a go at these exercises as a way you'll get the most out of this book, by trying yourself to apply some of the principles.

If you've just started at a new company, I encourage you to find time with a more senior tester and go through what you've done, and see if they have any feedback, especially ideas for anything you might have missed, or for a different way of doing an action. If not, find another person who's learning, and share your ideas, debriefing each other. Either way, you get to benefit from another's viewpoint and experience - typically you'll come to find there's an awesome power to collaborating with people in this way.

To the test leader/mentor/manager

So you've got a newbie starting next week, and you don't have anything prepared to help develop them through those first few weeks? Well, this is the book for you! I advise you to read through it yourself first, and encourage you to follow up on some of the exercises with your new starter. It should take them a couple of days to work through the book and the exercises, dependant on how intensively you need them to go through it.

This book is free, but if you do find it useful and valuable, I just ask that you contribute to the spirit of *"pay it forward"* that this book is all about. Consider making a charity donation from either yourself or your company. Iâ€™ve previously asked for donations to the childrenâ€™s healthcare charity The Starship Foundation[1].

[1] https://www.starship.org.nz/

Chapter 1 - Why does the world need testers?

As human beings we're incredible, imaginative, creative beings. We've made art, built huge buildings and visited every planet in our solar system.

But there's another side to being human, and that's we're not as infallible as we'd like to think. We might know basic arithmetic, we might know the difference between their, they're and there - but even so, once in a while our concentration will be off a little, and there it will be. *A little "oops"*.

In software programming, a simple grammar mistake of using a ":" instead of a ";" can cause problems. If we're lucky the code will simply refuse to build. If we're less lucky, that mistake will go unnoticed until that line is executed, and even then may pass unseen.

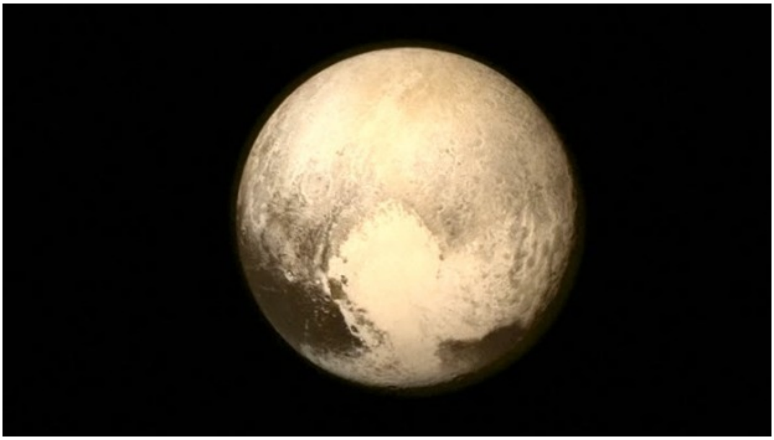
One huge problem is we're really poor at noticing our own mistakes. And this is compounded in software by an effect called *"confirmation bias"*.

Confirmation Bias

Let's just take a moment so you understand what confirmation bias is. There's a really good example on the recent New Horizons flyby of Pluto which I was guilty of.

I hold a Bachelor of Science in Astronomy, so as you can image, was getting somewhat excited at the initial pictures we got whilst the probe was still some way off. I knew from my degree how prevalent cratering is through the solar system - we see it most clearly on the Moon, where the lack of any weather system preserves these features almost indefinitely. But they're there on pretty much every solid body we've been to.

When I saw some of those early pictures of Pluto then I saw, without a doubt vague outlines of craters. Only there was a problem, after close encounter, everyone was talking about the strange lack of craters. How had I got it so wrong?



NASA Image

Confirmation bias works like this - if you look at a picture like that, which is slightly fuzzy, if you know that craters exist all throughout the solar system. Then you are going to see shapes which your mind will recognise as craters. Your mind is looking to confirm what it already thinks is there. The problem is, there are no craters there, you've created a mirage of geology in your mind, and you've fooled yourself. And this in many ways is how much magic and illusion works.

Confirmation Bias In Software

If a programmer has been working on a piece of software for a couple of days, they will have invested their time and energy on every aspect they know of resolving the problem. But this means when they come to test it, they won't bring any new ideas to the table. They will test it using the scenarios they've planned out in their head for it to operate for.

Like me and my craters on Pluto, they're expecting to see it work, so they'll only notice it failing if it's really obvious. Some bugs though can be more subtle, and more importantly can only be seen when trying different scenario.

Testing has developed as a career path because we've found some people are very good at using software, trying scenarios and describing problems they encounter. Often these people aren't developers and share very few core skills with developers.

In many ways it's the same as how an author for a book will need to use a proof reader for their material who will look at it with fresh eyes, and challenge any areas which don't make sense to them.

[Which reminds me – if you do find any obvious mistakes in this book – do drop me a line]

Chapter 2 - The test spectrum

There are a lot of testing types that are bandied about in the industry. This book is really focused on manual testing skills - this is a kind of testing where you as a tester are given a piece of software and you try certain actions on it, by typing or using your mouse yourself. It's typically the entry level role most testers will encounter.

These additional roles are discussed briefly here to allow you to understand the scope of testing which is typically performed within many IT companies.

Manual testing

You are manually entering a scenario into the software, *hence the term*. You do the text entry, the mouse clicking, everything. You are also responsible for noticing if anything *out-of-the-ordinary* happens.

You may be following a set of instructions - which is typically called scripted testing. Or else you might be following a general goal when tested - a broad umbrella of what's called exploratory testing.

Automated Checking

There are some forms of software which can be programmed to apply certain actions to a piece of software and to check for a predetermined response. Some people will refer to this as automated testing, but the circle of testers I belong to prefer the term automated checking.

Automated checking typically runs a lot faster than manual testing - but it takes a lot to script out (typically in a programming language), and it can only check for a limited set of (preprogrammed) behavior - which is the reason the term "check" seems more appropriate. They also tend to need maintenance when software is changed. Hence they're useful, but also have their limitations.

Smoke Testing

When software is deployed to an environment, typically a quick, high level test is run on it to make sure that everything seems to be working. Typically it's a test which quickly covers an end-to-end scenario - so for instance if you had a version of Amazon to test, you'd want to quickly confirm that you could,

- look for a book
- add it to your cart
- pay using a credit card
- receive an email of your purchase

Security Testing

This is a specialist form of testing - it attempts to review the system architecture and see if there are any ways in which your system could possibly be hacked or misused.

Performance Testing

This uses many similar features to automated checking. However here it’s used to simulate a large number of users accessing the system. This kind of system is only of value when your test environment uses similar hardware as your production one, and allows you to work out if there is a level of load where problems start to occur because either the system can’t keep up, or the user experiences excessive waits.

Chapter 3 - The heuristic test model

So here’s the core of what this book is about - how do you educate someone who is new to testing how to come up with test ideas?

An introduction to oracles and heuristics

The Testing Mindset

Typically when a developer writes an application, they will run a couple of checks on the software, it looks okay. QED: *“it works”*. If they’re writing for instance a registration page, they’ll enter their name and details (it’s the details which come to mind easiest for anyone), it processes their details, and they summarise that ***it works for me***.

For testers, this is a good start, but just because it works for *some*one, doesn’t mean it’ll work for *any*one.

The heart of being a tester is being able to ask “what if” questions – and for that scenario, testers would currently ask a barrage of questions,

- What if my name contains more than letters – for instance if the name is double barrelled or has an apostrophe?
- What if my name is really long?
- What if I’m really old or really young?
- What if my birthday is 29<sup>th</sup> February?
- What characters should not be allowed in my email address?

- What if I use a weak password?
- What if I leave a key field blank?

The answers to these questions form a series of tests or experiments that the tester chooses to run on the system, to find out places where the system behaves in a way which doesn't make sense.

Defining testing

We'll use the following definition for testing to guide us in this book – there are a lot of definitions out there, and I encourage you to read them all, and expose them to your scrutiny. But for now,

***Testing is about applying variety to a system in order to find problems in code*.**

Part of the core skills of a tester though is knowing "which variety" to apply. Testers who have been around a while, tend to guide what they choose to test from problems either they or their peers have experienced.

Obviously this makes things tough for a new tester, as without that experience to guide them, where do you start? A good first point is to have some exposure to heuristics – which are essentially rules of thumb which people have collected from testers experiences of *"things which often go wrong"*.

The rules of this section

There are three fundamental aspects which are the core of most online systems today,

- registration

- log in
- account self-management

Doesn't matter if you're Amazon, Facebook, Twitter or Hotmail, these elements are going to be within your system somewhere. So we're going to go through and explore each one, with a list of how I'd approach testing any such system.

For this section, I'm going to use as the core system under test Twitter, so if you've never used Twitter, now's a good time to get a handle. Now obviously some screens will change once this is published, but it's likely the core basics will stay the same *(so roll with it).*

Oracles

As I don't work for Twitter, so I have no idea of what requirements they have for their processes. Thankfully though requirements aren't the only way to take understand how a system should operate. The Context Driven School of Software Testing calls anything that can guide you to an understanding of the system an oracle, after the ancient Greek seers.

For my team, we define an oracle as a "guide for understanding how the system's supposed to behave". Whilst it's true a requirement is an oracle, they're not the only ones.

An oracle thus is something that also helps you to determine if a test has passed or fail. A standard kind of oracle is *"if I do everything a page instructs me to do, when I submit, it should allow me to proceed"*. If it asks for additional information which was not clearly marked as mandatory, then there could be a problem.

Let's think about an item we'll look at in the future regarding *"logging in"*. You enter your username and an incorrect password.

I have no requirements here - but what do *you* think will happen next? *[No, I'm not going to say until a future installment - but really hope you've guessed]* And how do you know you're right?

In part because you've used other systems with these core elements in, and expect the behaviour from one to be pretty close to the behaviour from another. And partly because it makes a kind of sense. If you can log into an account with any old password *(oops - spoilers?)* - then really, what's the point of your login screen?

Heuristics

Together with some oracles to guide us, I'm going to use heuristics to generate test ideas. Heuristics are rules of thumb used to test an element of a system. Typically for instance when you see a name field, you'll be generating test ideas such as,

- what if I leave it blank?
- what if my name is REALLY long?
- what if I've got apostrophes and double-barrelled elements to my name?
- what if my cat runs along the keyboard and it's a jumble like 89dee8(#EH#(*E? *[If you look at the pattern those letters make across the keyboard, then it's likely the cat was really breakdancing]*

All these test ideas are called heuristics, they are rules where you know systems can get into difficulty. So they're a good idea to test – because you expect problems there, and as a tester you're actively looking for problems.

Occasionally I meet a worried customer who'll desperately ask me *"your testing will include some negative testing, won't it?"*. It's the sign of a customer who has had someone work for them going *"I log in with my name and password … the login page is now tested"*, it means that the tester is likely to have not used enough heuristics to generate test ideas.

And let's be frank, sometimes you don't have enough time to apply every test heuristic, but you should try out as many as you have time for, often starting with the ones you know are likely to cause the most trouble!

Many of us experienced testers use what I like to call an "experiential heuristic", which is a fancy word for "we like to try and reproduce bugs that we have seen in other systems". Thankfully people like Elisabeth Hendrickson have collected together a whole load of people experiences together into what she calls a *heuristic cheat sheet*[2] as the source for a few ideas … if you've never come across this page. *I promise you, it's not cheating!*

Her sheet contain a lot of useful ideas to apply to test scenarios, and we will be exploring the next few scenerios using it to guide us.

You can find it at: http://testobsessed.com/wp-content/uploads/2011/04/testheuristicscheatsheetv1.pdf

Or Google *"Test Heuristic Cheat Sheet"*.

Testing registration

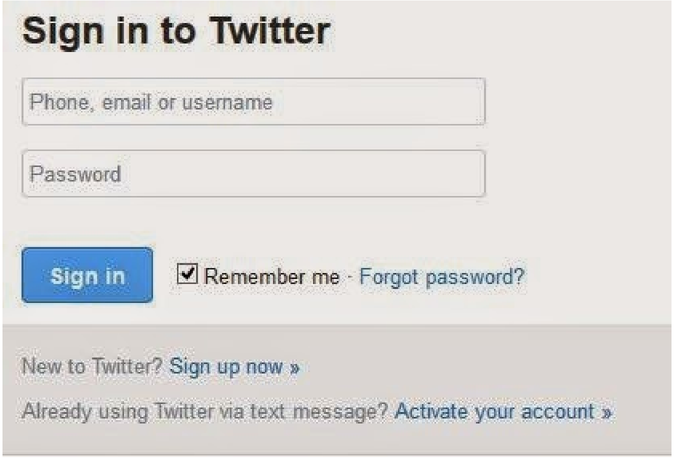
Next I'm going to look at how to apply some of these heuristics on a system which is common to most web applications … Registration.

Registration is the first time that a user encounters your system - make it too difficult or painful, and they won't continue past this stage. As we discussed previously, most people - *and especially developers* - when faced with "testing" this page will just enter their own personal details, hit "create" and if it works, then job done.

[2] http://testobsessed.com/wp-content/uploads/2011/04/testheuristicscheatsheetv1.pdf

However there's certain things you'd want to try out beyond that. Having a system that allows you to login is okay - as long as you're going to be the only person who uses the system (unlikely). As a tester your mission is to try out a representative sample of scenarios which will cover all the other potential genuine and malicious users.

Lets start by looking at Twitter's current registration page …



Sign in to Twitter

New to Twitter? Sign up now »

Already using Twitter via text message? Activate your account »

Twitter registration

Oracles

As discussed we don't have any requirements from Twitter, but having done this on other systems, we have a good indication of expected behaviour, and this experience can form a powerful set of oracles. Let's write out some of the basics,

- The system should set up a brand new account when done *(but we won't test this fully until later)* and an email sent to user
- My email/username should not have been used by anyone else before.
- The terms and conditions should be clearly readable

- All fields are mandatory - if not entered, no account will be created
- Email field must be valid according to rules – I've used a Wikipedia page on email format to help me, and that forms another oracle
- Password will need to be of a minimum complexity (but exact rules unknown)
- The text you enter into the password field should be obscured to prevent someone from leaning over your shoulder and seeing it
- If the system doesn't like what I'm doing *(breaking any of the above)*, it should at least fail gracefully with an error message that gives me meaningful information

Not at use here, but what I'd expect on this kind of page,

- I need to confirm I'm above a certain age *(usually by entering your birthday)*
- I need to agree to terms and conditions before creating my account *(here they have put that "by selecting 'create my account' you are agreeing to our terms and conditions)*

Picking out heuristics

From Elisabeth Hendrickson's heuristic cheat sheet, areas which seem to make sense to apply here are,

- strings - most of the fields are strings to enter, so most of the areas there apply to each field
- date and time - would apply if we had a date of birth field *(we're going to pretend there is)*

- boundaries - would apply for date of birth if we had a minimum age *(again, going to pretend there is)*
- input method - makes sense to ensure we can copy and paste to fields

What we're going to do now is to combine these to generate test ideas. *Don't be afraid to try them out with me.* Here goes ...

Test 1: Looking holistically ...

We're going to look initially at the system holistically, and then break down to be put each element under duress.

Let's start off with a few really basic tests ...

- Enter your details, including all the fields, confirm that account created *(might need to check the database for this)* and potentially an email is generated when account created. According to the CRUD heuristic, you might also like to once it's created to retrieve the account in some way. We'll cover some of this more in a future section, but if you have access, you might check in the back end to make sure the data entered is really that the system stores *(not just for this test, but for other variants where an account is created).*
- Enter absolutely no data, and try and create. Expect an error message to be created. *[On Twitter, it actually 'wags' the 'create my account' box at you to show an error, but close enough]*
- Create all the registration information, bar one field which is kept blank. Try to create. It should warn you.

Test 2: Name field

Okay so it can work for some people from those tests. Let’s try out some other options. We’re going to exercise the name field under duress now.

The most obvious things to apply here from the cheat sheet are strings and input methods. So ...

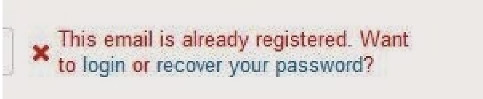
- We’ve tried it being blank already. But let’s make it really long, and see what happens. There are some tools out there to make really long strings. But I like going _“abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMONPQRSTUVWZYZabcd...“_. Typically eventually the field just stops, and I remember what letter it stopped at. I then create the account, and retrieve the name _(either from the account or by looking in SQL)_ and check it stops at the same letter, ie the name hasn’t been unnecessarily truncated
- The field “the quick brown fox jumps over the lazy dog” is helpful as it includes all the letters of the alphabet - if you use this and it’s returned in full, then there’s no crazy truncation of letters or after spaces. It’s useful to also include in UPPER CASE.
- Not all names are alpha characters. Some people have an apostrophe or insist on having a double barrelled name. And yet their names are valid names. How does your system handle that?
- Some people will insist on being French or German. Yes there is a world of accented characters out there - àáâãäåçèéêëìíîðñòôõöö - how does your system cope?
- Can you copy and paste your name into the field?
- How does it handle characters you don’t ordinarily associate with names? Things like numbers or special characters like _‘123...’_ or _’!@#*$(%*‘_. Eliminating these characters from entry into your system helps security - particularly the * is used in Unix and SQL can can cause all kinds of mischief.

- If my name is entered and including spaces, make sure that “Mike Talks” is retrieved as *“Mike Talks”* and not truncated on the space down to *“Mike”*.
- What happens if I start my name with a SPACE or use multiple spaces?

Test 3: Email field

As with the name field, you can try really long fields, copy and pasting etc. But there are also special rules for email fields.

A lazy *(but useful)* basic test would be to try the valid email addresses found under the Wikipedia entry about “valid email formats” including some examples of invalid address. Or at least try your own variants.

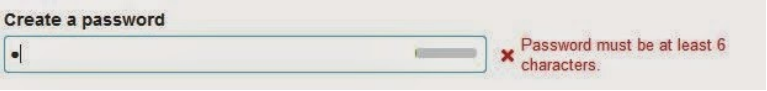


✖ This email is already registered. Want to login or recover your password?

As mentioned in the oracles, if you include an email address previously used, you’d expect some kind of warning.

Also would expect some “case insensitivity” in the checking of email. If I’ve used an email *“testsheepnz@tester.co.nz”*, I don’t expect to be allowed to use *“testsheepNZ@tester.co.nz”*.

Test 4: Password field



Create a password

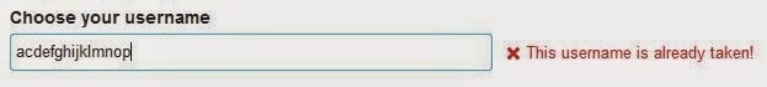
•

✖ Password must be at least 6 characters.

As said, blank should be unacceptable. Typically as mentioned in oracles, passwords have to be longer than 6 characters, and include not just lower case alpha, but some upper case of numeric/special characters are mandated.

Tests for passwords which are too short or too long should be considered, or those which break the rules. Also include copy and pasting passwords in.

Test 5: Username



Choose your username

acdefghijklmnop

✖ This username is already taken!

Again too long/too short variants again need to be used. If you try to go too long it should warn you, and you should try out using numbers and special characters to see whats allowed and what's discouraged.

As with email address, if you try to use an account already used it should warn you. And likewise employ a case insensitive check to prevent you from creating *"testsheepnz"* when there is already a *"TestSheepNZ"*.

Test 6: Date of birth (bonus feature)

Okay, Twitter doesn't have date of birth, but a lot of places do.

To throw some very basic spanners in the works, you should try and enter,

- A completely garbage date, confirm the system warns you it's not a valid date

- Include a date in the future for date of birth
- Try the 29th February for some valid/invalid leap year options
- Try some dates that don't exist such at the 30th February or the 32nd September
- Try first/last valid dates in the months
- Try entering dates in different date format - ie 12/01/94 or 12 01 1994 etc, to see how the system copes
- If the system asks you to select a date from a GUI, try different options, including selecting days that aren't there.

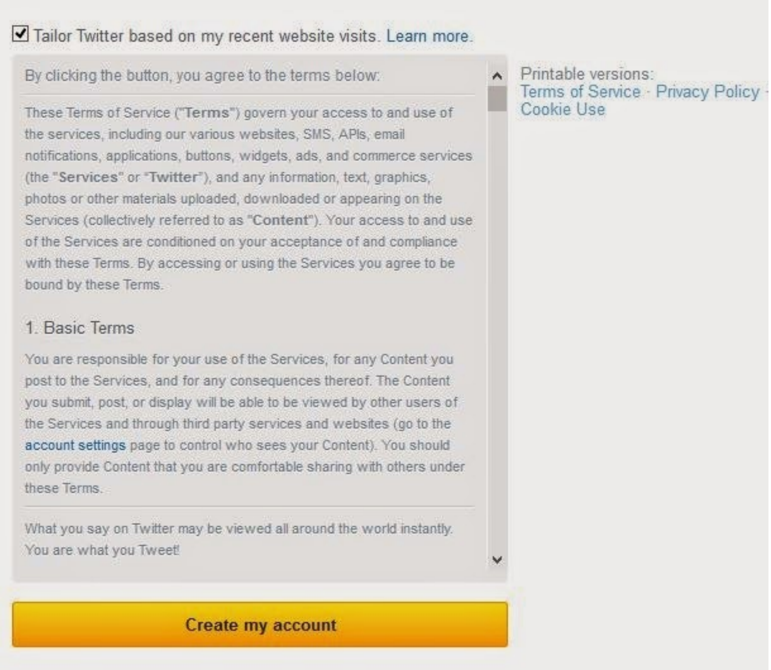
Typically most systems have business rules that "you need to be 13 or more to sign up for an account". This is a great time to try some boundaries,

- Try someone who is minimum age to the day
- Try someone who is minimum age minus one day
- Try someone who is minimum age plus one day
- Try someone who is over 100 years old. Pre year-2000, many computer systems stored date of birth year as a 2-digit field. So people over 100 sometimes cause a bit of a fit in legacy systems.

Test 7: Terms and conditions

Yeah, most people just go *"yadda, yadda, yadda"* for the terms and conditions. Typically you should not be able to create an accoutn unless you select you're accepting the terms and conditions. But also they need to be displayed well.

On Twitter it doesn't seem to be very well displayed, until you click in the box, and it expands like so ...



☑ Tailor Twitter based on my recent website visits. Learn more.

By clicking the button, you agree to the terms below:

These Terms of Service ("**Terms**") govern your access to and use of the services, including our various websites, SMS, APIs, email notifications, applications, buttons, widgets, ads, and commerce services (the "**Services**" or "**Twitter**"), and any information, text, graphics, photos or other materials uploaded, downloaded or appearing on the Services (collectively referred to as "**Content**"). Your access to and use of the Services are conditioned on your acceptance of and compliance with these Terms. By accessing or using the Services you agree to be bound by these Terms.

1. Basic Terms

You are responsible for your use of the Services, for any Content you post to the Services, and for any consequences thereof. The Content you submit, post, or display will be able to be viewed by other users of the Services and through third party services and websites (go to the account settings page to control who sees your Content). You should only provide Content that you are comfortable sharing with others under these Terms.

What you say on Twitter may be viewed all around the world instantly. You are what you Tweet!

Nice!

Wrapping up …

As you can see just from this very basic pass, we have created a whole ream of test ideas for a very basic operation.

You can see how a lot of testers *(especially context driven ones)* will look at these ideas and prefer to either have a check list of them or mind map. It's going to take a lot of time to write those all up as formal scripts, but you probably have a good idea how to execute them right now *(which makes one wonder 'so what benefit exactly do I get from scripting?').*

In fact, have a go at trying a few tests on the Twitter registration right now. Then take a look at the registration pages for Facebook, Hotmail, Instagram - pretty much anything you can find *(that doesn't involve a legal obligation).*

Browsers variation

On my first pass of the registration page using Elisabeth Hendrickson's cheat sheet, there was something that I missed, because the sheet itself is missing it! The sheet was created back in 2006, and in that time, there has been a bit of change in how we use the internet. It's a fantastic example of how, if you just work from a list, you can miss something. So let me introduce ...

The device/browser heuristic

Yup - back in 2006, IE was the dominant browser, and there was a wee upstart browser called Firefox which a lot of geeks raved about. But generally it wasn't considered to be too important or mainstream.

Welcome to 2015 - we have a host of browsers such as Chrome, Firefox, Safari. And we still have IE, but which version is important - *IE6?, IE11?, every possible version?*

Do we need to test on Windows 8, Windows 7, Vista? Do we need to test on Apple and Linux machines?

And that's not even going into mobile devices or tablets? Are we going to choose out a few versions of iOS or Android which are important?

The best thing to do is to work out which are the most currently popular devices out there - even better, ask marketing for what their target market is. Don't trust to devices you have in your testing pool to be representative of your market! *[And that market pretty much changes every time Apple makes an announcement]* A good place to look through is *Synapse Mobile*[3] who have links to what are the most popular mobile device platforms at present.

I never got to meet him, but when I lived in Farnborough, Stephen Janaway was just down the road from me testing for Nokia, and he has some excellent insights into mobile testing.

When you're looking at cross-browser testing (considering you've narrowed down your mobile and browser options to a suite to test on) is,

- Are all my fields present?
- Can I enter data/select items as expected?
- Is information displayed to me consistently?
- Does it look ugly and unusable? *Sometimes alignment can be all over the place, and it makes the screen look ugly. And something that's ugly often undermines trust in the legitimacy of the website.*
- How does the screen look on different resolution settings?
- How does the screen look on maximum vs resize of the browser? Or when I minimise the page?
- Does the description of the page on the tab bar make sense?
- What happens if I use the back/forward/refresh buttons?
- Do all error messages get displayed appropriately?

Playing around with mobile browsers, several interesting additional areas I've found are,

- What happens if I click a drop down box? Apple and Android have their own built in methods for dealing with these, and sometimes it can't handle drop down boxes with large amounts of text well.

[3] http://www.synapsemobile.co.nz/items.html

- How does the mobile browser handle being turned sideways from landscape to portrait and back again during an operation?
- How useable is the keyboard in landscape and portrait mode?
- Apple and Android don't really have pop-up boxes. Does your system have any? How does it handle them?

Most of these test ideas will of course apply not just to the registration but all the pages ongoing! We'll also take some time to look in more detail at the subject of browser testing later on in the book.

User login

One key heuristic you hear in use a lot is CRUD or ***C***reate, ***R***etrieve, ***U***pdate ***D***elete. This is the standard lifecycle of a lot of items you can create in a computer system.

In many ways an account is no different, we can

- create - when we register for an account
- retrieve - every time we login to the system
- update and delete - when we need to modify our account (we'll do that next)

So we're really going to do the CRUD testing over the whole section.

Oracles

Occasionally I come across someone who'll ask about testing the login *"I enter my username and password, how hard can it be?"*. As a great philosopher of modern times often says "D'oh!".

Really our oracle is not only *"how we've seen login systems work before"*, but also our understanding of why the login exists. The login exists to allow me access to my account, but to also keep others out. *Thats why we have the password!*

So any testing needs to be around the rules of when I should be able to log in, and when I shouldn’t. With computing power being what it is now, we don’t today have systems that allow unlimited tries to login. Typically there’s a rule out there that if someone forgets their password a set number *(typically 3-5)* consecutive times, then the account is locked. This stops someone just nuking the account with password attempts until it breaks open.

Locking can mean different things - it could be the user has to contact the helpdesk to get it unlocked. It could be it remains locked for an hour *(limiting the hacker to 48 attempts to crack the password a day, not great but still better than unlimited). But however, it fundamentally limits the number of tries you make, because any successful login after so many fails will be treated as a fail.*

Also we need to think of not just login but also logout. You want the system to auto log you out after inactivity. You don’t want to be able to logout, then someone log you in again by just hitting the back button. If your login url is *“yourapp/login”* and your account url is *“yourapp/account?userID=1234*” you don’t want to be able to get into the account just by changing *“login”* into *“account?userID=1234”* on the browser address bar. *[You may laugh, but it’s been done]. *

I have to admit the last two there *“back button doesn’t relogin”* and *“changing address bar to hack in”* are blurring the line of understanding that I have between an oracle expected behaviour and a heuristic. But in a way, it doesn’t matter - you don’t get exam questions on this - the important thing is that it’s used to generate a test to try.

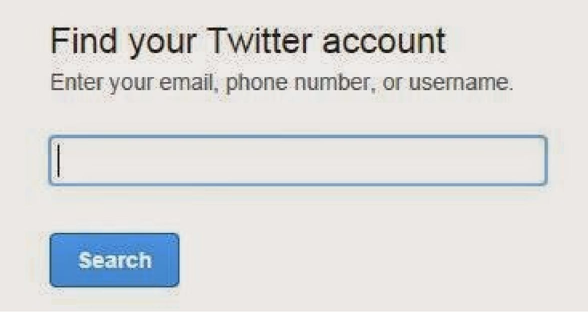
Finally, people do forget their password once in a while, there should be an option to say *“send me a temporary password”* via email to login to the system.

Oh - did I mention that the password should be obscured as you enter it so your neighbour can't just watch you enter it?

Heuristics

Actually not that many - there's just a few on-screen variables here. You should probably try that the maximum length of username entered matches the maximum length from the registration page. And likewise for the password field.

You should want to try a variety of character types used for password and username to check nothing's getting stripped or objected to.



Find your Twitter account

Enter your email, phone number, or username.

Test 1: Happy Day or *"I can login"* The following combinations will allow a user to login to their system,

- Enter their username exactly as entered into the registration page and with correct password
- Enter their username, but with different capitalisation and with correct password

- Enter their phone number starting 0064 for phone number area code and with correct password
- Enter their phone number starting +64 for phone number area code and with correct password
- Enter their email address exactly as entered into the registration page and with correct password
- Enter their email address, but with different capitalisation and with correct password

Should probably mention something about *"password is always obscured"*, but you surely have that by now? *If that requirement shocks you, you need to try out using more web systems for familiarity!*

Test 2: I can't login

The scenario above fleshes out a kind of requirement "when I enter my user nameand password then I'm logged in". That kind of scenario is often called "positive testing" (we're testing it happens). There are two variables we're providing here "user name" and "password" - what happens if one/both are invalidated? This kind of scenario is called "negative testing" (we're essentially testing a NOT case) - people will often ask if you're doing negative testing to confirm that you're doing more than just shallow testing an area of functionality.

It's possible to go even deeper than that though (although it's a good start). If the username/phone number/email address is entered with the following, the user is not let into the system *(with a suitable error message given)*,

- Incorrect password
- Correct password for another account
- A password which has since been changed *(old password)*

- The correct password but using different capitalisation
- The correct password but using the first *x* characters of an *x+1* character length password
- The correct password but using *x+1* characters for an *x* character length password (add a space at the end/start)
- A temporary password that has been used once.
- A temporary password, when another temporary password has been subsequently ordered.

If I give a username/phone number/email address for which there is no account, it makes sense the error message is suitably generic. Otherwise likewise, scammers/phishers can go *"this email has an account, but this one doesn't"*.

Test 3: Locked out

Lets say the lock out number is 5.

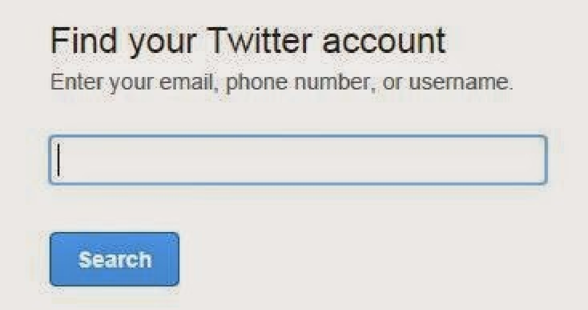
The basics

- I login with the wrong details for an account 5 times. I login with the correct details, but I'm not let in. [*It takes 5 consecutive failed logins to lock]*
- I login with the wrong details using a combination of username/email/phone number 5 times but incorrect password each time. I login with the correct details, but I'm not let in. *[5 consecutive failed logins against the account not the handle you're using]*
- I login with the wrong details for an account 5 times. I login with the correct details once and am let in. I login with the wrong details for an account 3 times. I login with the correct details once and am let in. *[It takes 5 consecutive failed logins, not 4 .... and they must be consecutive]*

Trying out the unlock feature

- I login with the wrong details 5 times. The account is unlocked. I can login. *[I can unlock my account]*
- I login with the wrong details 5 times. The account is unlocked. I login with the wrong details for an account once. I login with the correct details once and am let in *[Once locked once, it doesn't trigger for every failed login since]*
- I login with the wrong details 5 times. The account is unlocked. I login with the wrong details for an account 5 times. I login with the correct details, but I'm not let in. *[Locking the account isn't a one shot deal]*

Test 4: Forgotten password



Find your Twitter account

Enter your email, phone number, or username.

Lets order a new one! Click the button, to get the above page.

- You should get a new password via email if you enter, the email / phone number / username there. And by *"you should"* I mean you are going to test all those options ... *not just one of them! *
- An unused email / phone number / username version of any of them should be rejected

- Email and username should probably be case insensitive (*miketalks* being treated the same as *mIketALKs*)

If you've ordered a temporary password,

- If you use it to log in you are successfully logged in but made to change your password immediately
- You probably should still be able to use your old one, and if you login with the old *(good)* password, it should make the temporary password unusable *(try it)*
- You should be able to copy and paste the temporary password from the email into the screen
- There should probably be a time limit on how long the temporary password can be used for. *And guess who's going to test it?*
- If you order a temporary password twice, the first one should be made unusable. And hint - the temporary password should be unguessable, not *"newPassword01"*.
- You can't use a temporary password to login twice

Test 5:Logout When you are inside the system, you are logged out,

- After a set period of inactivity. *Stopwatch ready, because guess what you're doing.*
- If you close the tab/window, and then reopen it, you're taken to the login window to find you're logged out.
- If you select the "logout" button.

You cannot use the back button to get back into the system once logged out. Neither can you just type in the address bar for your account and bypass the login page.

Two tier variant

Some systems have what's called a *"2-tier login"*[4] system, where you use a password, then a token is sent to your mobile device *(or from a token system)*, which you have to use. This like a temporary password has one use, a new one can be requested (invalidating the old one), and has a limited time for which it can be used. All variables which can be tested.

The principle is it's safer to login using a password *(which could be broken)* together with an item you're known to have on you *(token or mobile)*. If your system has an SMS stub, you're in luck, and you can use a variety of mobile numbers, but still read the token details from the raw SMS stream.

If not, then it means sadly you only ever have one phone number you can use - your own. Suddenly that suite of phones for testing seems a good idea.

Observation

There's a heck of a lot to test there. It took me about 90 minutes to flesh all that out to that detail. Even so it's been a lot of typing, and my fingers are somewhat sore. To script it up in full (to a button-press by button-press level) would be at least 3 days, and for not much more benefit.

Now your turn

[4] http://en.wikipedia.org/wiki/Two-step_verification

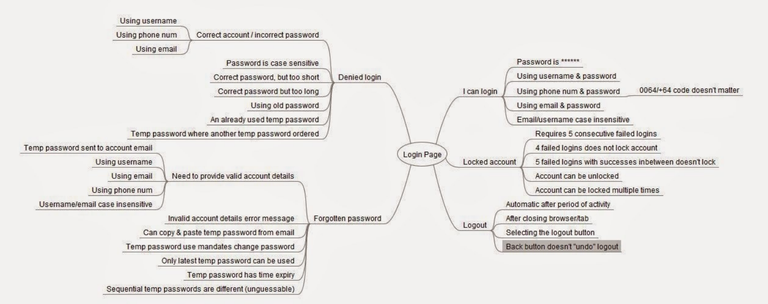
Have a play with your account, and again try this out on Facebook, Hotmail etc. Get a real feel for the behaviour, and as always comment below if you think there's anything you've noticed or would like to add!

Representing scenarios as a mind map

Before we plough on, I'd like to take a moment to talk about faster methods of representing data.

Within Wellington, Aaron Hodder is a particular champion of using mind maps to representing test planning, and has led frequent workshops and presentations on the subject.

The last section on testing Login took about 90 minutes to write out - I managed to represent the same data in a mind map in about 10-15 minutes. And what's better is that if you can master the mind map process, people tend to read these maps much more readily than a long set of scenarios.

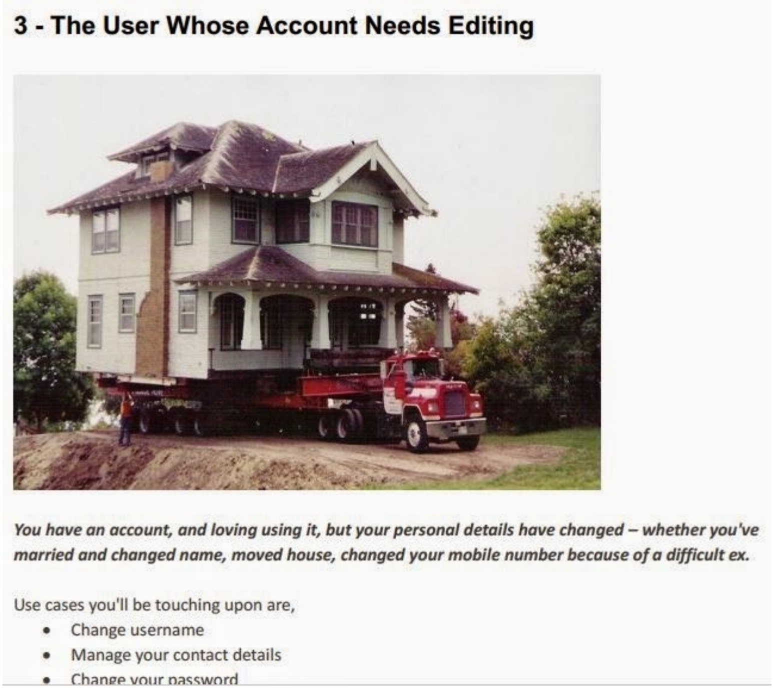

Login Page
Denied login
Correct account / incorrect password
Using username
Using phone num
Using email
Password is case sensitive
Correct password, but too short
Correct password but too long
Using old password
An already used temp password
Temp password where another temp password ordered
Forgotten password
Need to provide valid account details
Temp password sent to account email
Using username
Using email
Using phone num
Username/email case insensitive
Invalid account details error message
Can copy & paste temp password from email
Temp password use mandates change password
Only latest temp password can be used
Temp password has time expiry
Sequential temp passwords are different (unguessable)
I can login
Password is ******
Using username & password
Using phone num & password
0064/+64 code doesn't matter
Using email & password
Email/username case insensitive
Locked account
Requires 5 consecutive failed logins
4 failed logins does not lock account
5 failed logins with successes inbetween doesn't lock
Account can be unlocked
Account can be locked multiple times
Logout
Automatic after period of activity
After closing browser/tab
Selecting the logout button
Back button doesn't "undo" logout

Login test mind map

Account self-management

I hope the last few sections have made sense so far, because now we're going to put you into the driving seat a little more.

Now we're going to jump in with some oracle ideas about account self-management. Note that some of these ideas as well as *"testing expectations"* could also form *"design ideas"* as well … ***Oracles*** At my team we have a series of character cards and one of them is as below, "user who needs their account editing" …



3 - The User Whose Account Needs Editing

You have an account, and loving using it, but your personal details have changed – whether you've married and changed name, moved house, changed your mobile number because of a difficult ex.

Use cases you'll be touching upon are,

- Change username
- Manage your contact details
- Change your password

This is a range of functions which allow you to change some of the details of your account. The bottom line for a company, it really helps if certain very simple functions a user can modify for themselves over ringing a support line for. It's easier for them, and means you're not wasting money on call centre staff to do basic jobs. If your idea of service is making a customer wait 30 minutes on the phone whilst listening to Cat Stevens for a password reset, and more than likely the end result is your customer deciding they don't really need your system that badly.

Account Hijacking Danger!

There are some account details which if changed can lead to account hijacking. This is where if someone uses a machine in at an internet cafe after you and you're still logged in, they could potentially "hi-jack" your account. We'll talk a bit about those when we encounter areas where it could be dangerous.

Obviously a lot of the below depend on context of *"what service you are delivering"* and the level of rigour you need around it. Let's look at some details you might want to change ...

Changing Name

People change name a lot. The most obvious one is after marriage when it's tradition the bride change her surname to the groom's.

But there can be other reasons as well - for instance I have to admit not being too in awe of the surname *"Talks"*, as my school life pretty much consisted of every teacher saying at the start of the year, *"Michael Talks ... I hope he doesn't"*. I thought I had it bad until at University I met a guy called Nicholas Lunt who'd wanted to be a teacher, but decided otherwise *"because of what my name rhymes with"*.

Now if you're Twitter or Facebook or any social media, this should be a fairly easy thing to do. However I do know some social media make have safeguards - you can't for instance change your name to the name of someone famous like *"Kate Middleton", "Arnold Schwarzenegger" or "Donald Trump"* without flashing some ID to prove that's really your name. Facebook also has an issue with the surname *"Talks"*, which it thinks is so rare *"it's just a joke"*. Meh, thanks. So my son had issues creating an account in his real name – in the end he created one called "Zulu Warrior", which it allowed.

The difference for this would be if you have an online bank. For that given the ability for fraud by just changing your name, you'd want some more rigour and a *"come in and show proof of name change"*.

Changing Date of Birth

As far as I know, there is no way you'd ever want to change your date of birth. Okay maybe you'd want to younger, but there's no legal reason. If your system offers you to, then this is a bit of a no-no. Sure admin might need to correct, but the user?

Changing Gender

We can get a bit schoolboy giggly about this. But having experienced the other side of this, and the difficulty of changing gender through friends like Violet, having the world recognise your new gender if different from birth gender is a big deal. People who do need to be treated with respect and compassion, and not the butt of a joke.

The laws in the UK and NZ recognise the right of people to legally change their gender to that which they associate with, regardless of birth gender, so typically your system needs to as well.

Changing Password (Hijacking Danger)

Yes, it makes sense to change passwords occasionally. But to avoid the hijack scenario, it's generally good to ask for the old password first to prevent anyone from doing it. It also makes sense to send an email/text to say the password has changed *(but obviously NOT what the new password is)*, just in case you go *"wait a minute, I did not change the password on this"*.

Changing Email Address / Phone Number (Hijacking Danger)

Your email address and mobile phone number are typically used in *"forgotten password"* scenarios where you say you've forgotten your password, and a temporary one is emailed to you. If an unscrupulous party sets the email to one they control, then they're just a step away from hijacking your account.

Hence it makes sense as for password to have this change password protected, but also for there to be a *"changed email address"* notification sent to the OLD email address. And similar for the mobile phone number.

Hopefully I don't have to explain why it needs to be the old email address to you - if not, have a good think about it!

Similar logic to this applies to changing your mail address.

Heuristics And Test Ideas

This is the last in a series looking at oracles, heuristics and test ideas. Our first exercise quite thoroughly set out the details for testing account creation. The second exercise, regarding logon gave some more room for you to try things out yourself.

This time around, with those examples in your mind, and our expectations above clearly set out, you should be ready to fly this one solo!

What I strongly suggest is you have a go at this for your chosen site, then as I discussed at the start of the book, find a more senior tester in your organisation to discuss it with, to show your approach and working out. If you are still stumped for feedback, you can always drop me a line on Twitter, and I'll take a look over it when I have time. But be aware, that will be on a timeline of my availability, which might not be ideal.

I do encourage you though to find seniors, mentors or just peers with whom you can have meaningful discussions on testing.

Chapter 4 - Observation skills

Of course, being able to come up with test ideas is great, and I'm sure that you're starting to get the hang of applying these principles. However all that ability to plan means nothing if you're not able to notice an issue when it's right in front of you.

Hence good manual testers work hard to develop their observational skills when they're testing. Obviously when planning out testing in the previous section, we were setting out rules "if I do this and this" and expectations "then I expect that …". Pretty much "if … then …" statements.

But a good tester has their eyes always open, and will know sometimes when you're testing one thing, you might notice certain things which seem odd. You might even abandon your original test to look into this. This combination of observation and curiosity is key to what makes a manual tester so good at finding issues.

It's also why many use the definition "automated checking" over "automated testing", because although automation can be really useful, it has no ability to apply observation and curiosity, and then explore. It's why automated checks are better used for existing functions in software over newly developed software, because automation will miss a lot of things in newly developed code. And it's newly developed code where typically on the law of averages, that we expect the most problems. It's an area of change - which means it has added functionality, but due to human fallibility, also has the potential for added bugs.

Hence observation is a really key skill - it's also a really difficult skill to develop within you via a testing book. [Although I have a workshop which helps to develop this skill that I've run for the Datacom Test Camp and Summer Of Tech's testing bootcamp]

Ironically, my wife (who is not a tester) is really good at observation - she watches and reads a lot of crime fiction, so she is always on the lookout for clues. So maybe a bit of Sherlock Holmes or Miss Marple could help?

Here's a couple of examples to get you thinking though.

Example 1 - Comprehension test

They say everyone has a book in them, and I hope to write about more than just testing. So here is a passage from novel I'm working on. What do you notice?

Edward learned back as the clock chimed midnight in the deserted café, looking at June. The waitress, a young thing who looked like she would rather be out with her friends that night came over to take they're order.

"Coffee? It's free refills ..." She said in a kind of disinterested way.

Edward screwed up his face. He hated the way everyone asked that, "I don't like coffee, never have. Just make it a English brakefast tea thanks. Lemon, not milk".

June looked up from her menu and smiled, "Just a coke please".

The waitress scrawled their order on her pad, collected the menus, and ambled away to the kitchens.

Silence. There argument in the car had been two hours ago, and Edward hopped the pit stop would clear the air, but it didn't. To break the awkwardness, he found himself looking outside at the kids playing on the sidewalk in the bright daylight outside.

June followed his gaze, roller her eyes, then returned her attention to her nails, using her emerry board to file down the edges with critical *precision.*

The waitress returned with their drinks, and Edward took his espresso, and leaned back sipping it. It was going to be one of those days for sure.

What did you think? What would your first bit of feedback be? The incorrect use of *"they're / there / their"*? The spelling and grammar mistakes?

I know people who will stop and make corrections as they go along, and they will repeat this process until they get to the end. Such small things can become a distraction, and such people will sometimes miss for instance that the clock talks about it being midnight, yet the kids are playing outside in daylight. Or that Edward makes a big deal about not liking coffee, then happily drinks an espresso … when he ordered tea.

Spelling mistakes are easy to fix. But character and situation inconsistencies make you want to ask the author what their intention was.

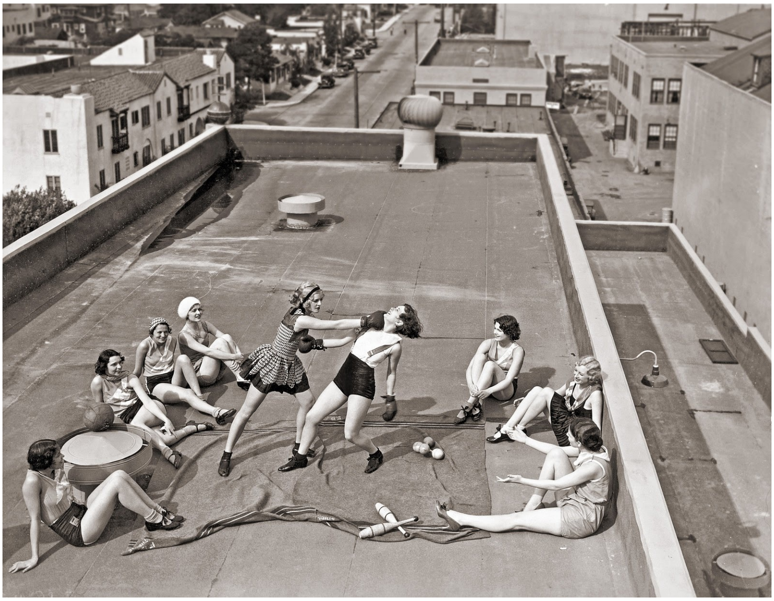
Of course, an important question you could also ask of the author is simply, *"apart from letting us know that June is pissed with Edward, is this scene supposed to serve any other function?"*. Of course literature doesn't have to serve a function, but it's worth asking the author why that scene's there, especially if it seems to go nowhere.

[Someone did get in touch with me, and mentioned "do you know how many spelling mistakes there are in the bar sequence?" You should have noticed it's the worst spelled piece in the whole book, and it's that way by design. I'm trying to use all those small problems to distract you from the fact that Edward says he hates coffee at the start, then happily drinks an espresso at the end. This is me trying to use something called perceptual blindness for my own means

It also means that if you see any mistakes in the book, you have to question if it's there because of human fallibility or because it's a *test from me. Srinivas who brought it to my attention acted like a good tester – he brought it to my attention, but know those mistakes were not likely to be a big deal, and "pull the book off the shelf until corrected"]*

Example 2 - Women on a roof

So what's going on here then?



You see this picture a lot on Twitter with a subtitle, which seems to explain everything "women boxing on a roof". So a 1930s fight club that no-one talks about then?

Take another look.

My boxing instructor Josette noticed something when I tagged her in this picture on Facebook. All the women are wearing dance shoes. There's also a trampoline, some juggling bats and balls there. Rather than proper boxing with that additional information, they look to be preparing for a show. And yet the first thing that grabs your attention is the two women throwing punches!

Now look at it again, and find things to confirm it's the 1930s, and not a group of modern hipsters using the black and white function on their iPhone. How'd you get on?

Practice opening your eyes and really looking at the world around you! As with the picture above, two women boxing are a magnet for the eyes. Get used to trying to look beyond *"the first thing you see"* occasionally.

Chapter 5 - Raising a defect

I was asked by a graduate tester about just what should go into a good defect report. It's a really good question, thinking about the information you provide when reporting a defect part of the key role of a tester. *After all, if we're not able to explain what we've noticed to someone else, that bug isn't going to go away any time soon!*

I've written many defect reports, but I've also been a developer who's fixed them. So I know what's helpful, and what's not. Something I really encourage new testers to read is a humorous meme that's out there[5]of supposed log book communications between pilots and ground crew on a fictional airline. The pilot is vague about the issues, so the ground crew are vague about their resolution back. *Don't let this happen to you!*

Problem: "Something loose in cockpit."

Solution: "Something tightened in cockpit."

Problem: "Target Radar hums."

Solution: "Reprogrammed Target Radar with the lyrics."

Problem: Suspected crack in windshield.

Solution: Suspect you're right.

Problem: Noise coming from under instrument panel. Sounds like a midget pounding on something with a hammer.

Solution: Took hammer away from midget.

As with most things - writing a great defect answers some very generic questions of ***What-How-Why-Where-When***!

[5] http://footflyer.com/Articles/JustForFun/AviationHumor/pilots_mechanics.htm

So let's start the ball rolling - you're using your companies new trial piece of software, and something just doesn't seem right about it. *Let's start defining it ...*

What

In a nutshell, ***what*** is the problem? This should be a real brief summary of what the problem is.

I like to think of it in terms of elevator pitches. Imagine you've just got into the ground floor elevator with your project manager. You get off at the 3rd floor, she gets off at the 4th. And she asks you *"hey, I heard you found a defect this morning?"*.

You have 3 floors, and about 20 seconds to summarise what you've encountered. This is a real skill in summarising what you've found to a one sentence tag. But believe me, defects like episodes of Friends[6]are remembered as *"the one with the..."*.

If you have a super and concise *"what the problem is"* summary, it probably belongs as your title. That way, anyone reading through your defect system will go *"ah"*.

Hint - when you go to talk to a developer about a defect you've logged, it's best not to quote just the number. very few people remember bug 666. That *"the one where ..."* summary will be the best way you have to job their memory!

How

Okay - you've summarised the problem. But ***how*** did you cause it to happen?

The how is a way of *"repeating your steps"*. Ideally with a defect, you'll be able to repeat a series of steps, and repeat the unusual behaviour.

Repeatability of a defect is great. But sometimes you just can't repeat the behaviour. *What then?* Well that's when you have to use

[6] http://en.wikipedia.org/wiki/List_of_Friends_episodes

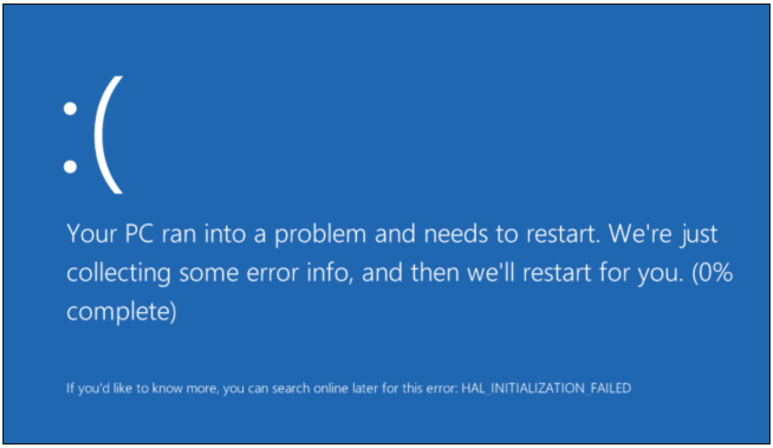
your judgement - often depending on how much of an issue you think it was. It’s always worth talking through with a developer.

This is one of the reasons I like to use a screen recorder when I can, because it records exactly what I did and where the issues happened. It is alright though to have defects that can’t be repeated, but it’s best to put on it*“I’m not able to repeat”*.

Oh - and they say a picture is worth a thousand words, so never underestimate the power of a screenshot!

Why

Why is this a problem? For you to be raising this as an issue you must find there’s something you don’t like about what you’ve seen.



:(

Your PC ran into a problem and needs to restart. We're just collecting some error info, and then we'll restart for you. (0% complete)

If you'd like to know more, you can search online later for this error: HAL_INITIALIZATION_FAILED

Sometimes it’s very black and white, *“the requirement says the page should display your account details, and instead it displays nothing”*. Likewise, if you encountered a blue screen of death, you can be pretty sure that’s not *“as per design”*.

But sometimes you might raise a defect because there is something going on which doesn’t feel right. It *bugs*you (another word we use for defect).

Your description of why will lead you to another description of a defect which is important - it's severity. Typically in projects there are more defects and oddities than time to fix. So people tend to focus on the things which are causing the most pain.

Severity is how severe a problem something is. If you're causing your machine to blue screen regularly, that's pretty severe, and going to impact what you can test. If you have a spelling mistake, that's less of a problem, and certainly not going to impact your testing too much. However as it doesn't take much of a spelling mistake to make a swear word *(as the test manager who emailed me with a mispelled request for "defect counts" found out).* And although it's true to say we testers live to report these kind of defects, they can actually have a functional impact - if you have a system which generates an automated email which includes a swear word, it's likely some e-mail filters are going to put it in the junk pile!

Generally though - although there are grids and standards for defect severity, I've found you just tend to pick this up through experience. In truth, if you get everything else right about your defect and leave the severity blank, most people can choose appropriately from the information you've provided. But experience helps you to find tune this.

Where and When

When it comes to retracing your steps ***where*** it happened and ***when*** it happened helps.

Where - well just that. You might have a couple of test environments, so it helps to know that. If you're on a web application, the kind of browser (and version) usually helps. And indeed the machine. All this information goes double for mobile devices of course!

When can be handy too - sometimes it's known for someone to be doing a release to an environment, and forget to tell testing.

Shocking huh - but it does happen. Or indeed it allows a developer to look through the logs around about that time to see if anything peculiar was going on in the logs.

Chapter 6 - Exploratory testing

For some of us old timers, we have a certain view of test practices which have been passed to us from the waterfall project roots we've all had some experience of. In many ways, these methods are familiar, especially to non-testers, and from this familiarity we feel a measure of comfort and security which sadly can often be unwarranted. One of the key ideas we tie ourselves in knots around is that "how will we know what to test if we don't write out test scripts?"

When veterans of waterfall such as myself are first faced with some of the concepts of exploratory testing our reactions can range from shock to abject horror. Almost always though the initial reaction can be a negative one.

This section will help to define exploratory testing, de-construct what we do in scripted waterfall delivery testing, and talk about how we provide the same essential value with exploratory testing. It will cover our approach to using exploratory testing as part of a system of testing, although of course other approaches can vary. This is how we've found to get not only the best results, but the best buy in and turned our detractors into our allies.

In Building A Testing Culture[7], an article based on conversations with other software testers, a major change and indeed the future of software testing was seen to be in the hands of increased use of exploratory testing. And yet the idea isn't new - it was first coined by Cem Kaner in the 80s. *So what is it, and why is it becoming increasingly applicable to the way we're running software projects?*

[7] http://testsheepnz.blogspot.co.nz/2014/12/building-testing-culture.html

A few homegrown definitions

Let's start by making a few definitions before we discover further! These are the definitions I'm comfortable with, but you may find definitions will vary.

A test script is a set of instructions and expectations for actions on a system which a user then confirms when executing testing

Historically I've seen many a project where such scripting has meant we'd define every action of every button press, and detail every response.

Typically in waterfall, we would have a good amount of time to test … *or rather we would ask for it, and whether we get it or not would be another tale.* A product definition would come out of requirements and design, and whilst everyone starts to program against these, testers start to produce plans of what is to be tested, coupled with defining scripts for the new functionality based on these definitions.

Once done these plans and scripts are ideally put out for review – if you're lucky you'd get feedback on them, which means you expand or reduce your coverage to include the areas which are key to the product.

When we run our scripts, typically life couldn't be simpler, all we do is go through, try each step, and tick or cross as we go to *"prove"* we've tested.

The problem with scripts is that they're very brittle,

- they require a lot of time upfront *(before the tester has access to a product)* to scope to the level of detail discussed above.
- they are fallable in the fact that just as developers are not immune to misinterpreting product definition neither are testers. *And that means large amounts of rework.*
- the requirements and design need to be static during the scripting and execution phase. *Otherwise again, you have a*

lot of rework, and are always "running at a loss" because you have to update your scripts before you're allowed to test.
- they depend on the testers imagination to test a product before they've actually seen a working version of it.

Unfortunately talk to most testers today, and they'll report to you these are areas they're under greatest duress. The two biggest pressures they feel revolve around timescales being compressed and requirements being modified as a product being delivered.

Some testers have a desire to *"lock down the requirements"* to allow them to script in peace. But obviously that involves to some extent locking out the customer, and although it's something that we might feel is a great idea, it's important to have an engaged customer who feels comfortable that they've not been locked out of the build process to have a successful project.

So testers have to be careful about not wanting to have a model for testing which works brilliantly in theory, but breaks down because of real and pragmatic forces on their project.

Exploratory testing has multiple benefits – one of the greatest being that it doesn't lock down your testing into the forms of tests you can imagine before you've even had *"first contact"* with a prototype version of software.

Exploratory testing is a method of exploring software without following a script. There are many ways to perform it – the best parallel I have found is with scientific investigation. You set out with an intention, and you try experiments which touch the area, devising new experiments as you go along and discover.

With exploratory testing there is more value in noting what you've actually done over recording your intentions. Compare this with scripted testing, where you put the effort in ahead of time, and all you record during your testing is either a tick for a pass, or cross for a fail!

A test session is a high level set of behaviour that the tester should exercise when executing testing. Often this can be a list of planned scenarios we think would be good to exercise. But unlike a script, it's a set of things to try in testing, without getting bogged down early on with the step-by-step instructions of how that will be done. It also is more a set of suggestions, with room for additional ideas to be added.

So what is exploratory testing?

There are many definitions out there for exploratory testing, and I'm going to add my own understanding of it.

Exploratory testing to my team is about

- building up an understanding first of what the core values and behavior of a system is *(often through some form of Oracle)*
- using that understanding to try out strategic behavior in the system to determine whether what we witness is unexpected

Skill Based Testing

This ability to think beyond just a one sentence requirement is part of the inherent skill which is a core need for exploratory testers, it calls for,

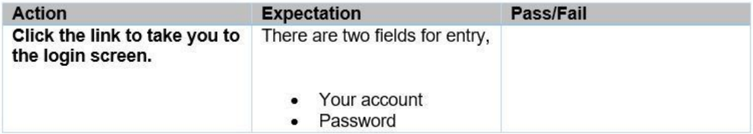
- An understanding of what the systems supposed to do. Not just functionality, but the "business problem" it's trying to address.
- An understanding of how the system has worked thus far is helpful
- An understanding of similar behaviour in other products

Exploratory testing is thus often referred to as *"skills based testing"*.

An argument often used in support of scripted testing over exploratory testing is that *"if you have your testing all scripted up – then anyone can do it"*. Indeed such scripts it's possible to provide to almost anyone, and they can probably follow them.

The problem tends to be that different people will interpret scripts differently. Most inexperienced testers will tend to follow an instruction, and if what they see on screen matches, they'll just tick a box, without thinking outside of it *(and hence would be quite happy with "you can no log in" example above because it was what the requirement said). Lets explore this with another example …*

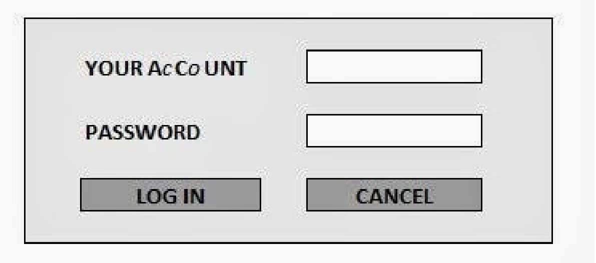
Test script

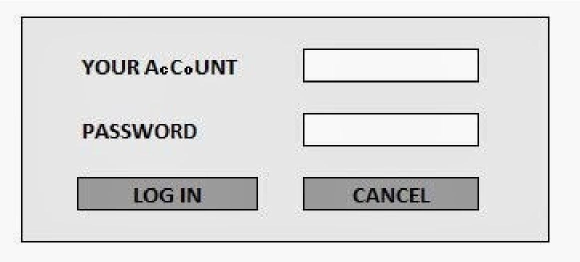


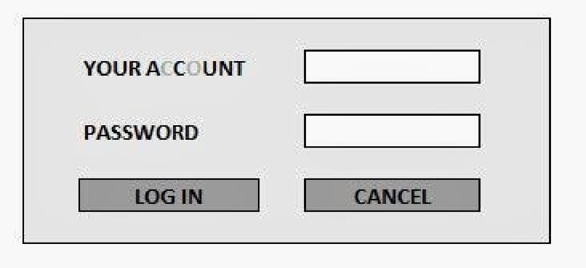
Action	Expectation	Pass/Fail
Click the link to take you to the login screen.	There are two fields for entry, • Your account • Password	

The Test System

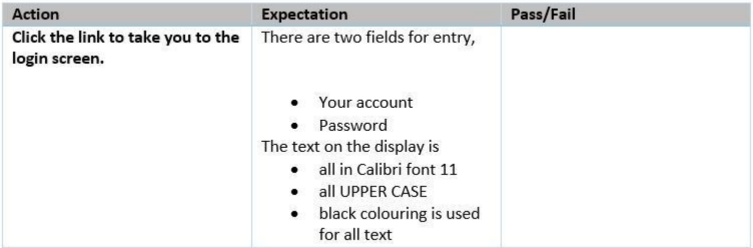
Here are 3 screens for the above script … do they pass or fail?







Can you be sure that every tester would pick up on that from the script? If so would it be a simple cosmetic defect, *yes? Sadly some people will combat this by trying to turn this into a version of the evil wish game, where you go overboard on explaining what you expect, so there' no room for ambiguity. Hence an obsessed scripted tester might try and write the expectations as,*



Action	Expectation	Pass/Fail
Click the link to take you to the login screen.	There are two fields for entry, • Your account • Password The text on the display is • all in Calibri font 11 • all UPPER CASE • black colouring is used for all text	

Well, that will stop those 3 problems above occurring again, but it won't stop other issues. Especially if it turns out the root cause for the problems is a disgruntled programmer who's tired of reading your scripts, and currently serving out their notice period until they move to another company where they do exploratory testing …

“But our scripts are our training!”

This is a common objection to exploratory testing. If you have detailed scripts, anyone can follow them, learn the system and be a tester right?

Most people I have spoken with have found that it’s far easier to have a handbook or guidebook to a product, which details at a light level how to do basic flows for the system, than to have high level of detail throughout your testing.

The counterargument to handbooks is that *“anyone can follow a script”* but as discussed previously to build up familiarity with the system and the values, you will always need some training time to get that. If you just throw people at the *“your account”* problem you’ll get differing results, you need people to be able to be observant and in tune with the system, and that doesn’t tend to happen when you just throw people blind at a system with only scripts to guide them.

The bottom line is that there are no shortcuts to training and supporting people when they’re new testers on their system. If you’re going to train them badly, you’re going to get bad testing.

One of the few exclusions we’ve found to this is areas which are technically very difficult - there are in our test system some unusual test exceptions we like to run, and to set up these scenarios is complex, and completely unintuitive *(includes editing a cookie, turning off part of the system etc).* This is one of the few areas where we’ve voted to keep and maintain scripts *(for now)*, because we see real value in the area, and with maintaining so few other test scripts, we can do a good job, without it chewing up too much time.

But we’ve found we certainly don’t need test scripts to tell us to login, to register an account etc - when the bottom line is our end users only get the instructions on the screen to guide them. *Why* *should a one off user not need a script to register, and yet testers like ourselves who are using the system every day require a script to remind us that if we enter the right combination of username and password, we're logged in?*

Debunking the myths

A common complaint about exploratory testing is that it's an ad-hoc chaotic bug bash chase. You tell a team to *"get exploratory testing"*, and they'll run around like the Keystone Cops, chasing the same defect, whilst leaving large areas of functionality untouched.

With such a view of exploratory testing, it's no wonder a lot of business owners see it as a very risky strategy. Whilst such ad-hoc testing can be referred to as exploratory testing – it's not an accurate or fair description of many people's experience of exploratory testing is like. People who talk about exploratory testing in such terms often damage the credibility of the technique, creating countless problems for its practitioners.

Just because exploratory testing doesn't involve huge amounts of pre-scripting, doesn't mean that exploratory testing is devoid of *ANY* form of pre-preparation and planning.

Indeed, when James and Jon Bach developed the idea of session based test management[8], it was all driven about making exploratory testing planned and accountable.

You will often hear the words *"session based"* or *"testing charter"* being referred to – these are a way at looking at the whole system and breaking into areas it's worth investigating and testing. The idea is that cover your sessions, and you will cover the key functionality of the system as a whole.

Drawing up a map of sessions

Creating a map of sessions is actually a little harder than you'd first think. It involves having a good understanding of your product and

[8] http://www.satisfice.com/sbtm/

it's key business areas.

Let's pretend we're working for Amazon, and you want to derive a number of sessions. Two key sessions jump out at you right away,

- Be able as a customer to search through products and add items to the basket
- Be able to complete my payment transaction for my basket items

Beyond that, you'll probably want the following additional sessions for the customer experience,

- Be able to create a new account
- Be able to view and modify the details of an existing user account
- Give feedback on items you ordered, including raising a dispute

Finally obviously everything can't be user driven so there is probably,

- Warehouse user to give dispatch email when shipment sent
- Helpdesk admin user to review issues with accounts – including ability to close accounts and give refunds

The trick at this point is to brainstorm to find the high level themes to the product. Ideally each session has a set of actions at a high level that really encapsulates what you're trying to achieve. For more on mind mapping *(I tend to be very much list driven in thinking)*, and I recommend Aaron Hodders notes[9] on the subject.

[9] http://www.assurity.co.nz/community/our-thoughts/part-1-aaron-hodder-on-using-mind-mapping-software-as-a-visual-test-management-tool/

Fleshing out the planned test session

For many skilled testers, just giving them the brief *“be able to create a new account”* will be enough, but perhaps a lot of detail came out of your brainstorming that you want to capture and ensure is there as a guideline for testing.

Let’s take the *“be able to create a new account”*, here are some obvious things you’d expect,

- Can create an account
- Email needs not to have previously been used
- Checking of email not dependent on case entered
- Password

  o Must be provided

  o Has a minimum/maximum length

  o Must have a minimum of 2 special characters

- Sends email recript

A session can be provided in any form you find useful – mind mapped, bullet point list, even put into an Excel sheet as a traceable test matrix *(if you absolutely must).* Whatever you find the most useful.

Such bullet pointed lists should provide a framework for testing, but there should always be room for testers to explore beyond these points – they’re guidelines only.

Keeping it small

Typically not all test sessions are equal – the test sessions for *“adding items to basket”*, if you’ve got over 10,000 items to choose from *(and no, you’re not going to test all of them)* is obviously a bigger task than creating an account. However if some sessions are too big, you might want to split them up – so you might want to for instance split *"adding items to basket"* to,

- Searching for items
- Retrieving details for items
- Adding items to basket

But really breaking it down is more an "art" than a set of rules, as you'd expect.

Test Ideas For Your Session

Okay, so you have a session, which includes some notes on things to test. Hopefully with your experience in the project, and test intuition you have some ideas for things to try out in this area.

A common thing testers use to explore and test is through applying heuristics, which we have focused on in such detail previously

But here are a few of those which should jump out at you,

- Boundary tests – above, on the limit, below
- Try entering too much/too little data
- Use invalid dates
- Use special characters within fields
- Can a record be created, retrieved, updated, deleted

Using heuristics, a simple statement like ***"can I create an account"*** can be expanded to be as large/small as needed according to the time you have.

Working Through A Session

In running and recording test sessions we've gone right back to University. For those of us who did science classes, an important part of our lab experience was our lab book. We used this to keep notes and results of experiments as we tried them out.

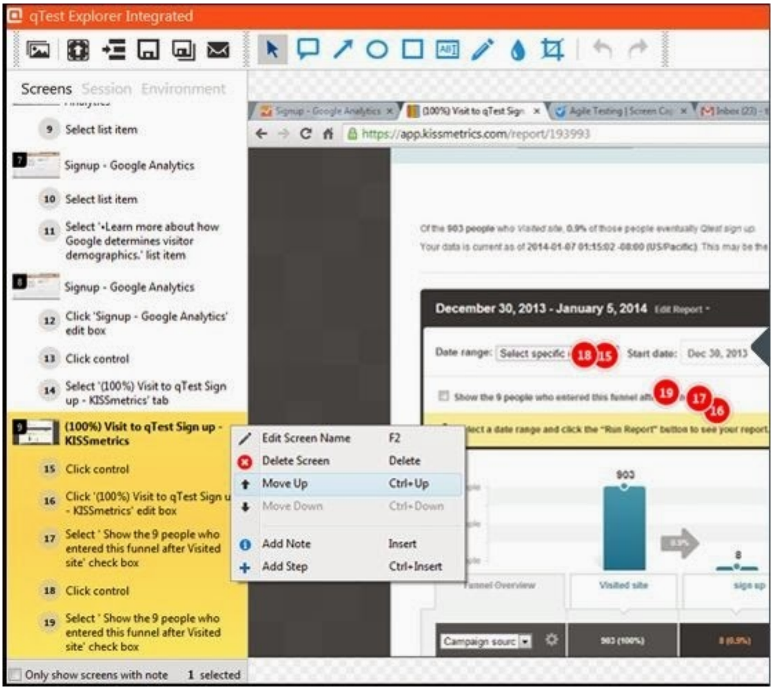
They weren't wonderfully neat, and we'd often change and vary our method as we went along. But we could refer back to them when we needed to consult with other students or our tutor.

For our group, these notes are an important deliverable as we move away from scripted testing. Typically we'd deliver our scripts to show what we intended to test. Now when required we deliver our notes instead showing what we tested *(over intended to test)*. Not every exploratory testing group needs to do this, but to us, it's an important stage in building the trust.

Obviously a key part of this involves *"just how much detail to put in your notes"*. And this is a good point. The main thing you're trying to achieve is to try our as many scenarios as possible on your system under test.

We've found a table in simple Microsoft Word is the most comfortable to use *(you can make rows bigger as you need to put more details in)*. Basically you put in your notes what your intentions are.

We couple this with using a tool to record our actions when we test. Each row in our notes table covers about a 10-15 minute block of testing we've performed. We've used qTrace to record our actions and the system responses during this time.



qTest Explorer Integrated

Screens Session Environment

9 Select list item

7 Signup - Google Analytics

10 Select list item

11 Select '•Learn more about how Google determines visitor demographics.' list item

8 Signup - Google Analytics

12 Click 'Signup - Google Analytics' edit box

13 Click control

14 Select '(100%) Visit to qTest Sign up - KISSmetrics' tab

9 (100%) Visit to qTest Sign up - KISSmetrics

15 Click control

16 Click '(100%) Visit to qTest Sign u - KISSmetrics' edit box

17 Select ' Show the 9 people who entered this funnel after Visited site' check box

18 Click control

19 Select ' Show the 9 people who entered this funnel after Visited site' check box

□ Only show screens with note 1 selected

https://app.kissmetrics.com/report/193993

Of the 903 people who Visited site, 0.9% of those people eventually Qtest sign up.

Your data is current as of 2014-01-07 01:15:02 -08:00 (US/Pacific). This may be the

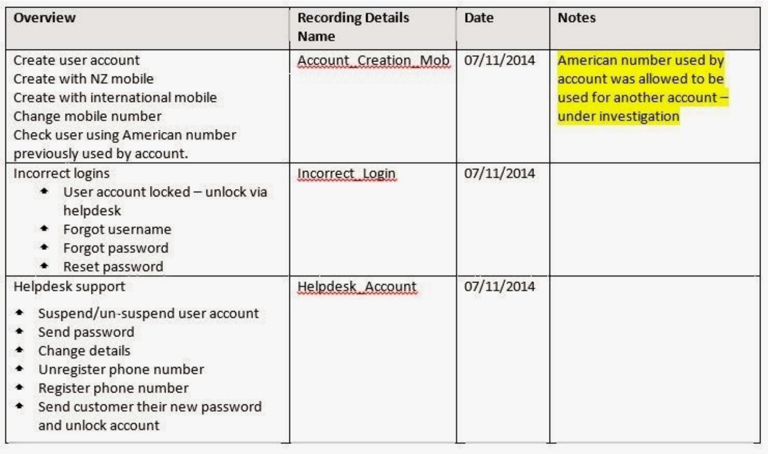
December 30, 2013 - January 5, 2014

Date range: Select specific Start date: Dec 30, 2013

□ Show the 9 people who entered this funnel aft

...lect a date range and click the "Run Report" button to see your report.

This allows us to use our notes as a high level index into the recordings of our test sessions. Obviously things like usable names for our recordings help – we go by date executed, but anything will do.



Overview	Recording Details Name	Date	Notes
Create user account Create with NZ mobile Create with international mobile Change mobile number Check user using American number previously used by account.	Account_Creation_Mob	07/11/2014	American number used by account was allowed to be used for another account – under investigation
Incorrect logins • User account locked – unlock via helpdesk • Forgot username • Forgot password • Reset password	Incorrect_Login	07/11/2014	
Helpdesk support • Suspend/un-suspend user account • Send password • Change details • Unregister phone number • Register phone number • Send customer their new password and unlock account	Helpdesk_Account	07/11/2014	

A screenshot from a typical session log

It should be noted of course that not all exploratory testing needs to be recorded to this detail. For my group, as we move away from scripted testing, this approach allows us to build up trust in the new system. The bulk of this documentation – screenshotting, and a button-by-button trace of what we do – we find qTrace invaluable in absorbing the load for us *(it would slow us down like crazy otherwise).*

Of course, any test we find ourselves doing again and again is a candidate for automation – and even here, if we have a suitable flow recorded, we can give a qTrace log to our developers to automate the actions for us.

Closing a session

When a tester feels that they've come to the end of a session, they take their notes to another tester, and discuss what they've tried, and any problems they've encountered. The peer reviewer might suggest adding a few more tests, or if there have been a lot of bugs found, it might be worth extending the session a bit.

This peer reviewing is essential, and we encourage it to lead to some peer testing. Basically no matter what your experience, there's no idea that you can have that can't be enhanced by getting the input from another individual who understands your field.

This is why for us, the method we've used for exploratory testing really confounds the standard criticism that *"when you write scripts, they're always reviewed" (even this is rapidly becoming the mythical unicorn of software testing, with many scripts being run in a V0.1 draft state).*

In both using reviewing *(through brainstorming)* of when we plan our test sessions and also using a level of reviewing when we close a single session, we're ensuring there's a "second level of authentication" to what we do. Sometimes we wonder if our process is too heavy, but right now an important thing is that the reviewing, being a verbal dialogue, is fairly lightweight *(and has a fast feedback loop)*, and both testers get something out of it. Being able to just verbally justify the testing you've done to another tester is a key skill that we all need to develop.

Peer Testing

We try and use some peer testing sessions whenever we can. I've found it invaluable to pair up especially with people outside the test team – because this is when the testing you perform can take *"another set of values"*.

Back at Kiwibank, I would often pair up with internal admin end users within the bank to showcase and trial new systems. I would have the understanding of the new functionality, they would have a comprehensive understanding of the business area and *day-to-day* mechanics of the systems target area. It would mean together we could try out and find things we'd not individually think to do. Oh, and having two pairs of eyes always helped.

Pairing up with a business analyst, programmer or customer is always useful, it allows you to talk about your testing approach, as well as get from them how they view the system. There might be important things the business analyst was told at a meeting which to your eyes doesn't seem that key. All this helps.

It's important as well, to try and *"give up"* on control, to step aside from them machine, and *"let them drive and explore a while"*. We often pair up on a tester's machine, and etiquette dictates that if it's your machine, you have to drive. It's something we need to let go of, with a lot of the other neuroses we've picked up around testing …

Chapter 8 - Browser testing

Browser evolution

It was 1991, and I was having a one-to-one tutorial with my cosmology lecturer, Professor Fred Combley, regarding a presentation on the Big Bang I was due to do in a few days. Behind him was a VAX terminal that he'd left on, and I could just about make out the word CERN on the screen from where I sat.

If you had asked me back then what on that screen would revolutionise the world, I would have guessed it was some results from a collision experiment that would change our understanding of the Universe. *And I'd be wrong.*

What was revolutionary what how the text had got onto the screen. Professor Combley was one of a team of nuclear physicists which included Doctor Susan Cartwright *(my other tutor who I'm still in contact with)* from the University of Sheffield. They were part of a huge network of physicists working on the Large Electron-Positron Collider[10] in Geneva. However being so geographically spaced, they needed something to help them to communicate and share information easier.

Tim Berners-Lee[11], a computer scientist and physicist who was working for CERN thought he had a solution. It was to provide a series of interlinked hypertext documents defined in a language called HTML1.0. These pages could be accessed via the internet using a program called a browser, which would render them onscreen. In his quest to make sharing information easier, he invented the very medium you accessed this book from, the world wide web.

[10] http://en.wikipedia.org/wiki/Large_Electron%E2%80%93Positron_Collider

[11] http://en.wikipedia.org/wiki/Tim_Berners-Lee

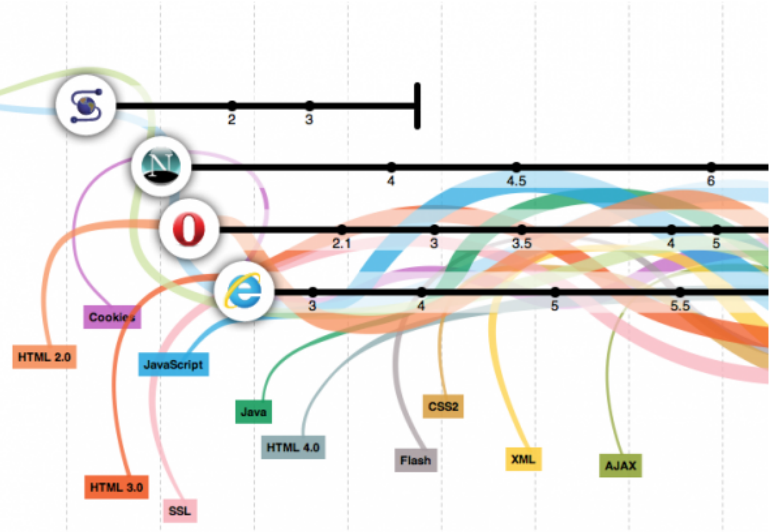
The world wide web takes off …

The usefulness of the world wide web soon became apparent, even outside of the world of physics. It was a phenomenon which soon snowballed over the next 20 plus years as more and more people got involved on the world wide web.

As the world wide web took off, there were two factors which really complicated the life of the modern day tester …

Factor 1: Everyone started to make browsers.

Originally it was Netscape, who built one the first browsers for UNIX machines. They were soon followed Opera, then Internet Explorer … *more and more followed.*



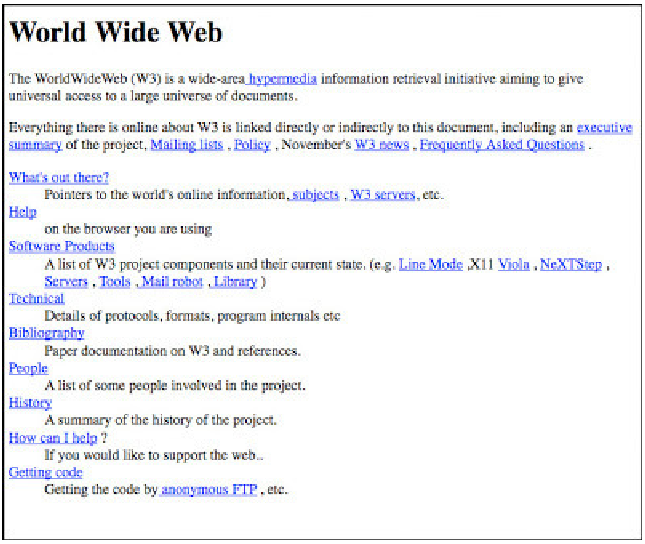
Mixed in with this, the different browsers treated HTML slightly differently. In 2000 I worked on a naval intranet project where a page was designed to read a table of the days events, and show these events as items on a 24-hour clock. The page looked beautiful in Netscape, but in Internet Explorer, which interpreted the screen co-ordinates differently for the events, it was ugly and messy. We had to put conditional logic into the page to work out which browser it was one, and treat the two browsers completely differently.

Many companies decided that rather than keep up, their customers could only use Internet Explorer (IE) for their website, this didn't seem too much to ask when most people were on a Windows variant, and 90% of people used IE anyway. But times have changed. People have Apple and Linux machines now, and have drifted away from IE in increasing numbers. And many potential customers have machines dating back a bit which don't even support the latest version of IE.

Upgrading the Windows license for your company is a big investment - many companies have so much software that works on a particular platform, that migrating is a major undertaking. Back in 2009, I was consulting at a company where our machines were still running NT - and was asked if I could test on Google Chrome (it's uninstallable on NT, in case you wondered).

Factor 2: It was just too darn successful

Take a look at the first web page ever published…



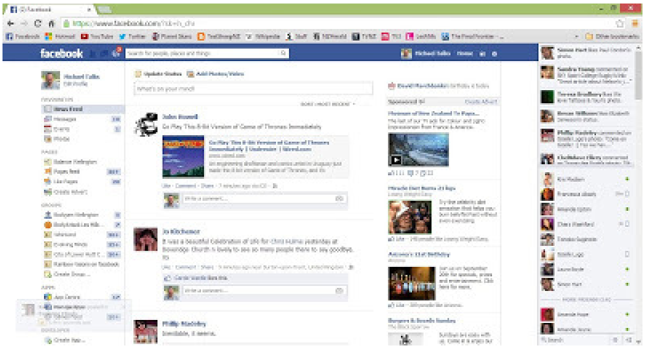
World Wide Web

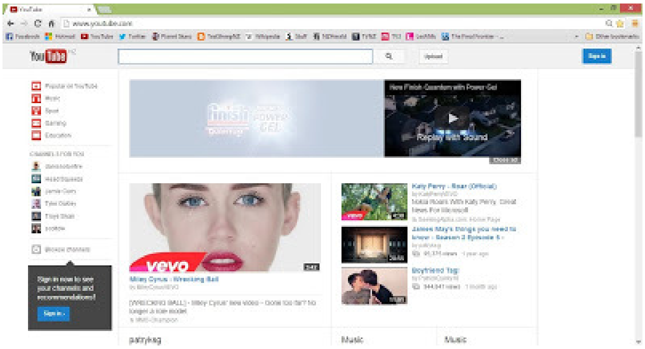
The WorldWideWeb (W3) is a wide-area hypermedia information retrieval initiative aiming to give universal access to a large universe of documents.

Everything there is online about W3 is linked directly or indirectly to this document, including an executive summary of the project, Mailing lists , Policy , November's W3 news , Frequently Asked Questions .

What's out there?
    Pointers to the world's online information, subjects , W3 servers, etc.
Help
    on the browser you are using
Software Products
    A list of W3 project components and their current state. (e.g. Line Mode ,X11 Viola , NeXTStep , Servers , Tools , Mail robot , Library)
Technical
    Details of protocols, formats, program internals etc
Bibliography
    Paper documentation on W3 and references.
People
    A list of some people involved in the project.
History
    A summary of the history of the project.
How can I help ?
    If you would like to support the web..
Getting code
    Getting the code by anonymous FTP , etc.

Today, it's barely recognisable as a web page. Now take a look at a couple of more modern examples…





When Tim Berners-Lee imagined the World Wide Web almost 25 years ago, modems worked at what we think of as dial-up speeds. He couldn't imagine being able to stream a video at dial up speed.

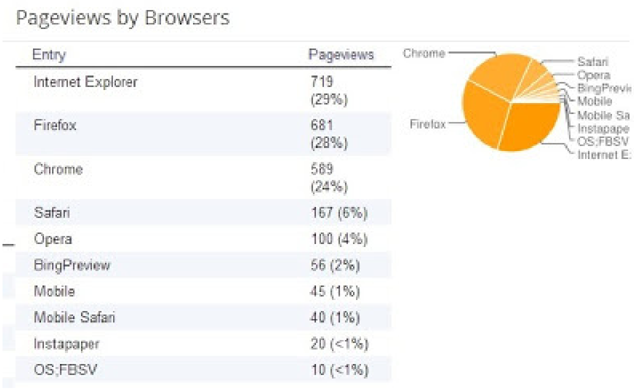
As the web became more popular, people pushed the definition of a web page, and it evolved. They started to include pictures, then audio and video as internet speeds increased. From being a single page of information, they started to be broken down into panels as peoples monitors got bigger.

As this happened, HTML, the language which made browsers possible, evolved - it's now on version 5. Java the programming language came about in 1995, and people started to embed some programming into the browser layer in Java, Perl and a variety of other scripting languages.

With what we're doing on browsers changing, naturally the browsers themselves are changing to evolve to this demand. Hence can the website you're developing today still be effectively rendered by a browser that's a version or two behind?

And this is the dilemma for the modern business - do you want to only reach potential customers with the latest/recent browsers? That's not a tester decision to make, it's not even a developer one. It's one that marketing people need to provide guidance for.

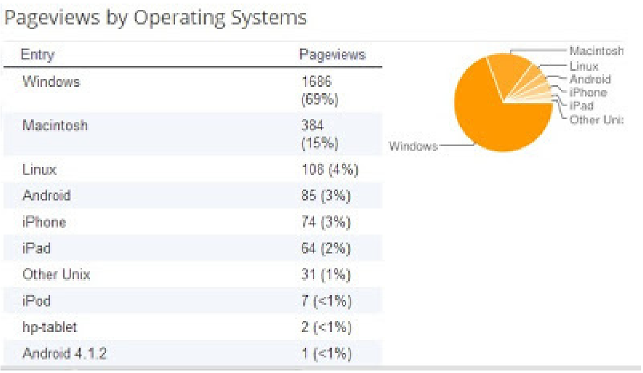
Fortunately help is at hand ... Most websites, if you already have one, can monitor the traffic they receive. This covers the page views I get on my blog by browser…



Pageviews by Browsers

Entry	Pageviews
Internet Explorer	719 (29%)
Firefox	681 (28%)
Chrome	589 (24%)
Safari	167 (6%)
Opera	100 (4%)
BingPreview	56 (2%)
Mobile	45 (1%)
Mobile Safari	40 (1%)
Instapaper	20 (<1%)
OS;FBSV	10 (<1%)

And by Operating System…



Pageviews by Operating Systems

Entry	Pageviews
Windows	1686 (69%)
Macintosh	384 (15%)
Linux	108 (4%)
Android	85 (3%)
iPhone	74 (3%)
iPad	64 (2%)
Other Unix	31 (1%)
iPod	7 (<1%)
hp-tablet	2 (<1%)
Android 4.1.2	1 (<1%)

Most companies don't choose to service everyone, but to make sure they're covering their core users.

From those stats above, I can see for instance, there's a good case for checking my page content and how it looks on IE, Firefox, Chrome, Safari, Opera. That would cover about 94% of my audience with 5 browsers.

Supporting cross browser testing

So you're aware some cross-browser testing is required, and marketing have come back to you with an extensive list … they'd like you to test,

- IE 11 (when available), 10, 9, 8, 7
- Chrome the last three versions
- Firefox the last two versions
- Safari just the latest version
- Opera just the latest version

Cool - let testing commence! *Only … it's not that simple.* There are some tools out there which will let you compare different versions of the same browser, and whilst better than nothing, they don't give you a full feel.

The most obvious issue is generally a computer has just a single version of a browser on your machine. *If you upgrade IE 10 to 11, it removes your version of IE 10!*

Likewise things get complicated because IE 7 *(for instance)* doesn't easily install on a Windows 8 machine, and needs a lot of trickery to get working. Likewise running IE 11 on an XP machine is never going to happen. For all these things, the situation created by hacking the system to make it work is likely going to invalidate and erode some of your assumptions for your test.

It's not surprising then in such an environment, the idea of doing browser testing by installing different versions of browser then stripping them off your machine is *non-too-popular.*

What has taken off to aid browser compatibility are virtual machines[12]. A virtual machine allows you to emulate another *(obviously less powerful)* computer within your own. It's run from a file which sets up everything about the machine, and means you can use your Windows 8 machine, but pretend to be a Windows 7, XP or*(shudder)* Vista machine. It's also heaps cheaper (but not as much fun) as having a whole suite of reference machines of different build. But then again, you can have dozens of them set up differently, running on one machine *(though not concurrently obviously … as "performance may vary")*. If you have Apple hardware, you can pretend to be a Windows machine *(but alas not the other way around)*.

Each virtual machine is a file which can be shared amongst testers *(providing they have similar machines)*, and each one can be set up with the appropriate platform and browsers you want to test on.

This means once your suite is set up, all you need to do is run up the machine you want - no continuous install/uninstall. You just keep and share a library of the machines, and boot up the one you need. No needing multiple machines. Another tester I was talking to told me with virtual machines, as long as you keep backups, you can try out crazy things and almost trash the machines trying different

[12] http://en.wikipedia.org/wiki/Virtual_machine

things *(including editing registry settings)*, because you can just return to the original *(unmodified)* backup so easily if you really get into trouble. It means you can test not just on different browsers, but represent different platforms as well.

Some of the most popular virtual machine tools at the moment belong to VMWare[13]. *Check them out.*

Obviously virtual machines aren't easy to set up, and need some tech buy in and support. However they're an increasingly popular method of having different browsers and machines "to hand" when testing.

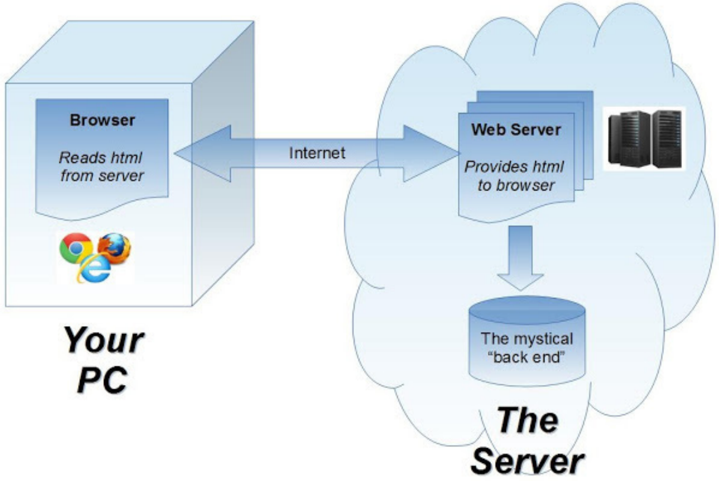
What to test when cross-browser testing

Taking a pure black box approach, if we have 11 browser types and versions we need to test comparability on, then that means we have to run the same testing we've got planned 11 times, once on each browser, yes? This is where such an approach becomes messy - testing is always *under-the-gun*, and such approaches just aren't going to work. To have a valid strategy we need to think about how websites work *(at a very basic level)*, the kind of issues we'd expect with browser incompatibility and go to where the risk is.

Web Basics

What follows is a really basic, ***"just enough"*** description of a web site, and how it works.

[13] http://www.vmware.com/products/player/



So, when you look at a page, your browser is provided information from a web server to create a page. This information is in the form of html, which essentially forms of *"downloadable program"* of content for your browser.

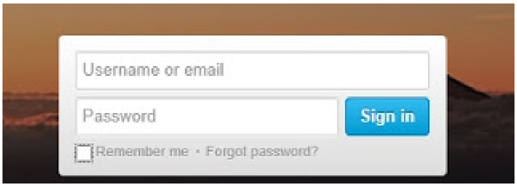
What exists on your browser is completely independent of the web server at this point. In fact, right now, disconnect your PC cable, or turn off your wi-fi. You'll find that this web page just does not simply *"vanish"* if you do. It exists on your machine, in it's browser, and won't disappear until you try and do something.

To get a good browser testing strategy, it helps to know what functionality is to do with what's on your browser, and what functionality exists on the part of the web server *(and it's back end).* And that can be actually harder than you'd think. As a user, most of our web experience is seamless between the two *(black box).*

But to be effective in browser testing, we need to be testing those features which are associated with the web pages rendering in the browser. But the features which are driven from the web server and the back end we're less likely to need to test so frequently *(the back* *end is the same back end, no matter the browser used).*

Example - the good old login screen

We're going to recap the Twitter login page here.



Here are some typical test scenarios you might explore

- ***"Happy Day"*** - there's a Username field, a Password field, and a Sign in button. When you enter the right username and password, you're logged in.
- If you enter the right email and password, you're logged in.
- If you enter the right email but the incorrect password, you're not logged in
- If you enter the right username but the incorrect password, you're not logged in
- If you enter the wrong username or password but the correct password, you're not logged in
- If you enter the wrong password for an account 3 times, your account might be locked
- If you enter the wrong password for an account 2 times, followed by the right you, you're logged in. If you log out, enter the wrong password again, followed by the right one, you are logged in, and your account is not locked.

Usually in a system that "*logs you in*", the functionality that decides from the account and password detail you've provided if you will either be logged in, not logged in, or your account locked all lives in the “web server” side of the system equation *(ie, not in the browser part).*

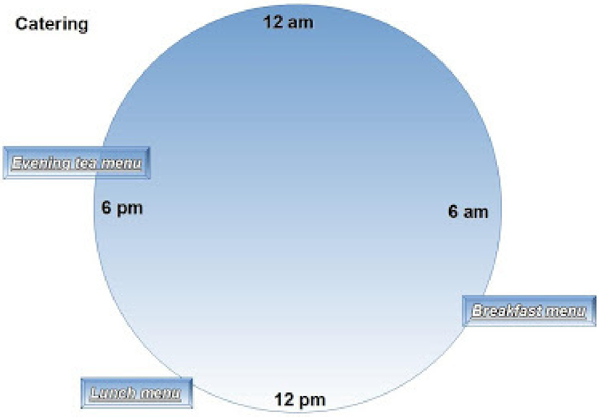
So for each browser you’d probably want to see the “Happy Day” path - looking at the page, and the basic flow through. There is also a case you might want to see the “*incorrect login”* and *“account locked”* messages at least once.

But all the validation rules listed above you expect to reside server-side. Which means you probably need to run them at least once if you can, but not for every browser - this might be a good assumption to make, write down and *“make visible”* to see if someone *technical architect-y* disagrees. This doesn’t mean it might not be worth running these tests again in different browsers anyway if you have time, but it means it looks to be *“low risk”* for browser issues.

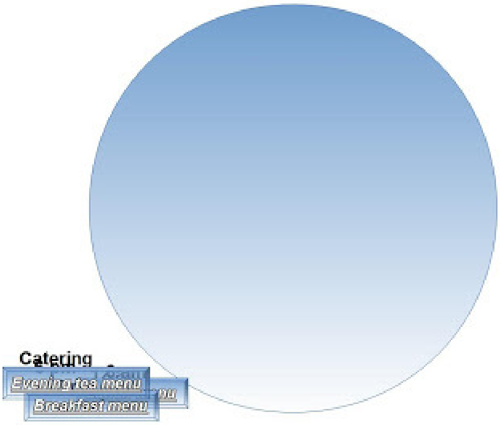
So what kind of issues should you be expecting?

Back in WEB101, I talked about an web application I originally wrote which didn’t cope well in IE.

In Netscape, it looked like this,



But in IE, all the labels got stuck in a corner, like this,



In web testing, we expect that there will be something in the html that the browser won't handle well, and hence interpret badly.

So for example - in the Twitter login example, we might be missing the fields for our username, password or the sign in button. *Those kind of details are cosmetic defects yes?* Oh .... except if these things are missing or hidden, we can't supply our username, password and select to login *(you can see how it makes an issue a lot more functional).* It makes our whole system unusable on the browsers.

The kind of issues we'd expect could go wrong with a browser, are roughly,

- missing/hidden fields and buttons - high impact
- script code for items such as drop down menus might not work - high impact
- buttons out of alignment / text out of alignment - low impact

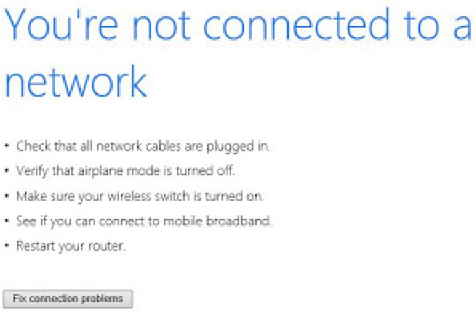
---

What behaviour is browser side?

We took an educated guess with the login example that everything about validation was web side.

I find the following rules of thumb useful for investigating which behaviors which are browser side. First of all, as mentioned above, everything you see on a page, has the potential to be browser specific *(so use your observation skills).*

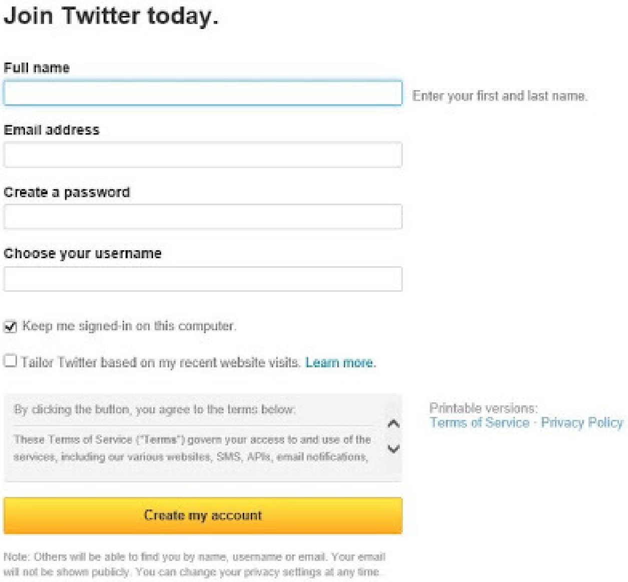
But secondly, for behavior and functionality, try this - unplug your machine from the internet *(physically or by turning off the wi-fi).* Anything you can do on that page that doesn't end in an error like below, indicates functionality which is browser side over web server side,



You're not connected to a network

- Check that all network cables are plugged in.
- Verify that airplane mode is turned off.
- Make sure your wireless switch is turned on.
- See if you can connect to mobile broadband.
- Restart your router.

Let's take a look at signing up for a Twitter account …



Join Twitter today.

Full name

Enter your first and last name.

Email address

Create a password

Choose your username

☑ Keep me signed-in on this computer.

☐ Tailor Twitter based on my recent website visits. Learn more.

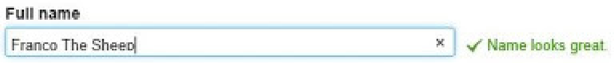
By clicking the button, you agree to the terms below:

These Terms of Service ("Terms") govern your access to and use of the services, including our various websites, SMS, APIs, email notifications,

Printable versions:
Terms of Service · Privacy Policy

Note: Others will be able to find you by name, username or email. Your email will not be shown publicly. You can change your privacy settings at any time.

I opened the page, then turned off my wi-fi. Let's work through a few of these fields …

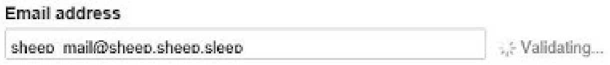


Full name

Franco The Sheep

✓ Name looks great.

Full name - we can enter anything, and it validates if it looks like a real name - all browser side functionality.

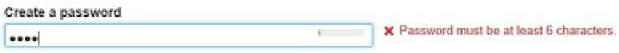


Email address

sheep_mail@sheep.sheep.sleep

Validating...

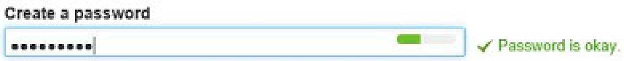
Email address - we can enter any text we like, so that's browser side. But it goes into a circular loop trying to validate the email address (I suspect to see if it's been used) - so that behavior is in the the web server.



Create a password

••••

✗ Password must be at least 6 characters.

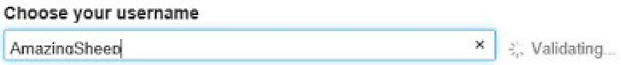


Create a password

•••••••••

✓ Password is okay.

Create a password - we can enter text, and it coaches over whether it thinks the password is long enough or secure. All this seems browser based.



Choose your username

AmazingSheep

Validating...

Choose your username - as with email, you can enter information, but it gets hung up validating. I think this again checks for uniqueness, and the validation is at the web server end.

Experimenting in this manner allows you to made educated guesses at the kind of behavior on a page that's browser based, and hence worth revisiting for different browsers when time allows. This gives you an educated guess at the areas of greatest risk in terms of browser compatibility. It's not infallible, but it gives you a decent rule of thumb to have conversations about what you think needs retesting a lot, and what needs less retesting with the rest of the team.

Chapter 9 - Testing responsive design on mobiles

Since it's introduction in 2011, responsive design[14] has become increasingly important in websites, and with good reason. Prior to it's use, websites struggled to support the increasing uptake of access from smartphones and tablets. Some sites would have to develop separate (and thus costly) websites - one for viewing from PCs, and one for viewing for mobiles (typically hosted at m.*).

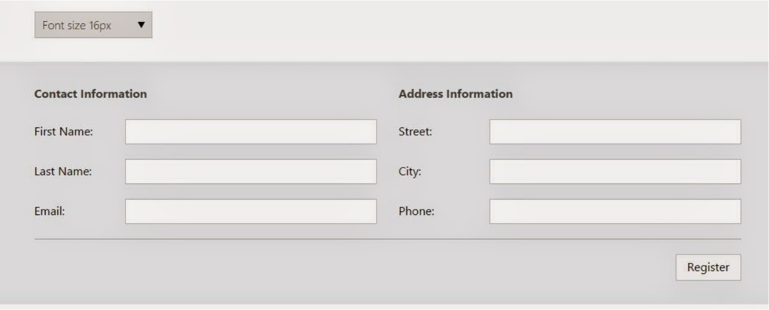
Responsive design was a methodology of having the same page source, which would scale according to the size of page available - allowing a single source of web content to be supported on difference sizes and devices - from laptop to smartphone. It basically allows all your information to "fit" horizontally, so if you're on a mobile device, you don't keep having to resize or scroll your screen horizontally <->.

There is a wonderful website you can try this out on the Telerick website under their demos[15] …

Open in full page on your browser, and it should look a little like this …

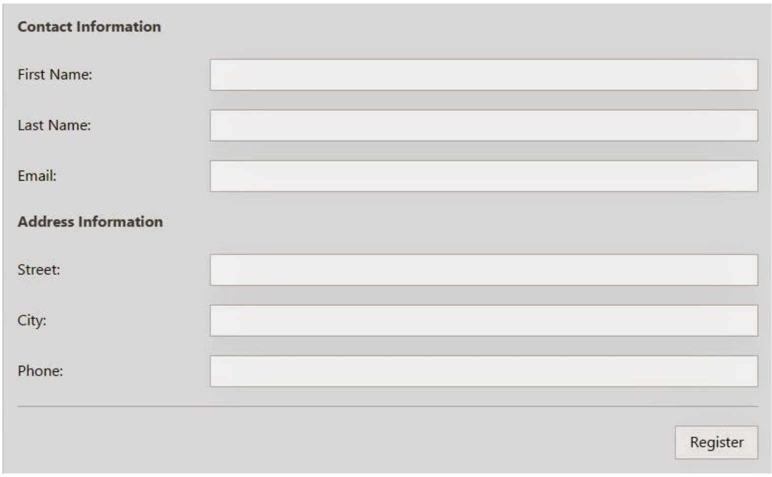
[14] http://en.wikipedia.org/wiki/Responsive_web_design

[15] http://demos.telerik.com/responsive-web-design-aspnet/samples/elastic/registration-form.aspx



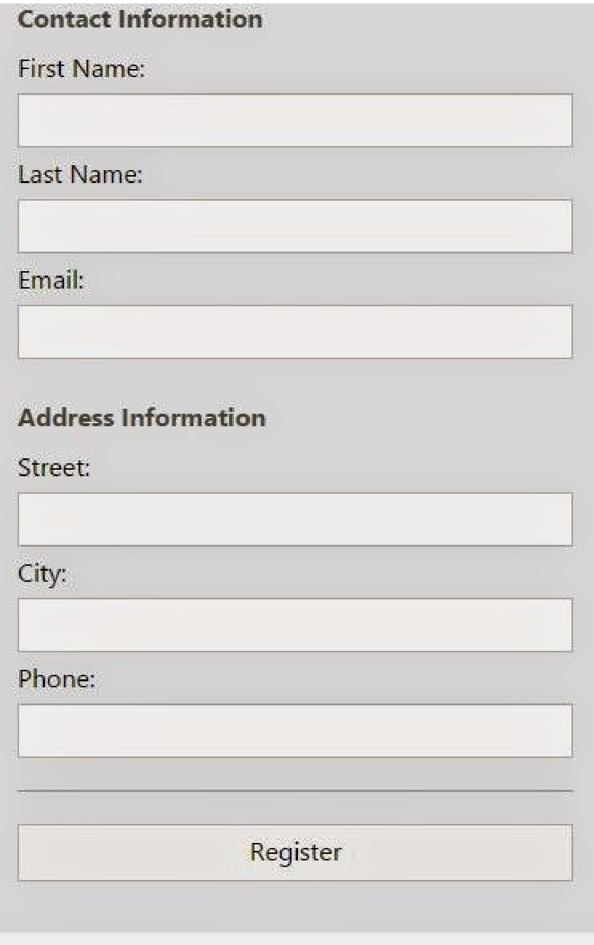
Telerick demo website

Now reduce the size of your browser a little, and you should find that instead of two columns of entries, it reduces to a single column ….



Telerick demo website

Now take it down even smaller, as if you have the limited number of pixels you get from a mobile, now the labels go above the data fields …



Telerick demo website

Pretty cool huh? But there are also a few potential pitfalls, and this article will talk you through some of them.

Whoo-hoo, mobile testing, I'll have me some of that!

Something we need to be crystal clear about, when we're talking about testing responsive design on mobile devices, we're basically just using them as browsers. This isn't connected with testing mobile applications which can be installed and work through Google Play or the Apple's App Store, that's a whole other different field of testing *(but some people get confused).*

Creating a strategy

To get the ball rolling, you need to start setting a strategy - across mobile devices and browsers. Responsive design uses some newer HTML features, so there are older phones and browsers which really struggle. So the question has to be - what browsers/devices matter?

When we've done browser testing in the past, we've just tended to install a whole host of browsers onto our machine, and maybe some virtual machines to cover older versions of IE, and just "get started". Here's the catch though - it's free to install a browser *(yes, even Safari).* But mobile devices cost - and if we're talking a high end model, then it costs a lot! You have to be selective, and your client has to be willing to support the purchase of these devices.

Even "hey guys, does anyone have a phone, I just want you to check this site in testing" is a bit dubious. You're running your testing strategy from *"what you can find around the office"*. The other thing is that a mobile phone is a very personal thing - I might check a site for you, but I wouldn't let you take away my phone to look at a problem I'd encountered.

If your client is keen on developing a responsive design site, then they need to be comfortable with renting or at least purchasing some devices. And here's the thing,

- Going forward, you will have to do some responsive design checking for every release. It's not just something you bolt on, work for a few week and don't have to worry about anymore.

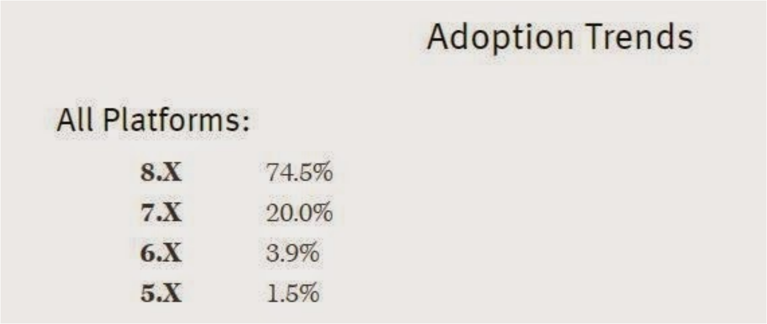
- New devices are always being released. This means revising those devices you use about every 6-12 months *(a good rule of thumb is every time there is a new version of iOS).*

Which devices?

The best answer to this is to talk to your client and ask "what browsers/devices have 5% of traffic or more".

Of course if you are just creating a responsive design, you might not be able to have reliable figures. In this case there are lots of sources out there. Mobile test consultancy Synapse[16] provide some useful resources (and I've used Jae before, and he's well worth bringing in).

Apple devices - you can find information about the most popular here[17].



Adoption Trends

All Platforms:

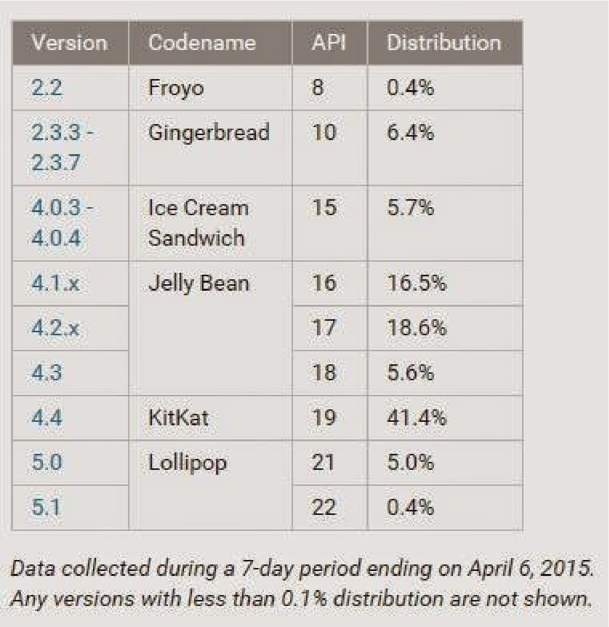
8.X	74.5%
7.X	20.0%
6.X	3.9%
5.X	1.5%

Android devices - you can find information about the most popular here[18].

[16] http://www.synapsemobile.co.nz/items.html

[17] http://david-smith.org/iosversionstats/

[18] http://developer.android.com/about/dashboards/index.html



Version	Codename	API	Distribution
2.2	Froyo	8	0.4%
2.3.3 - 2.3.7	Gingerbread	10	6.4%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	5.7%
4.1.x	Jelly Bean	16	16.5%
4.2.x		17	18.6%
4.3		18	5.6%
4.4	KitKat	19	41.4%
5.0	Lollipop	21	5.0%
5.1		22	0.4%

Data collected during a 7-day period ending on April 6, 2015.
Any versions with less than 0.1% distribution are not shown.

Right now, the jury is still out on Windows Phone 8.1 uptake, as is being able to cross compare device usage (iOS vs Android vs Windows 8.1).

Looking through that list, I'd say at a bare minimum, you'd need to consider the following in your test suite,

- iOS 8.X device
- iOS 7.X device
- KitKat device
- JellyBean device

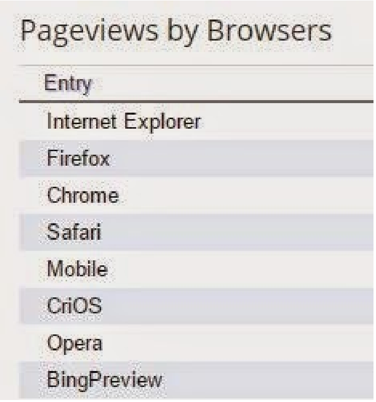
For these devices, I'd also try and consider lower spec models, especially with smaller screen resolution. [In responsive design, smaller screen means less real estate, and potential for problems. As testers we like problems] That can often mean looking at a smartphone over a tablet.

Beyond that, I'd try and see it I could talk up a purchase of something in Lollipop *(it's a low share, but it's the future)*, and maybe Windows 8.1 *(especially as there are some dirt cheap Windows Phones out there right now)*.

Regarding the browsers on those devices - most people just use the build in browser *(until analytics tell you otherwise)*.

Remember - this is my analysis from the current market - it will change! Once your site is up, try and get analytics to help profile popular browsers/devices, after all it doesn't matter what other people are using, what matters is what your client's customers are using.

And on that bombshell, just a few weeks after Microsoft announced the death of IE, look who sits at the top of the most popular browsers for reading this blog?



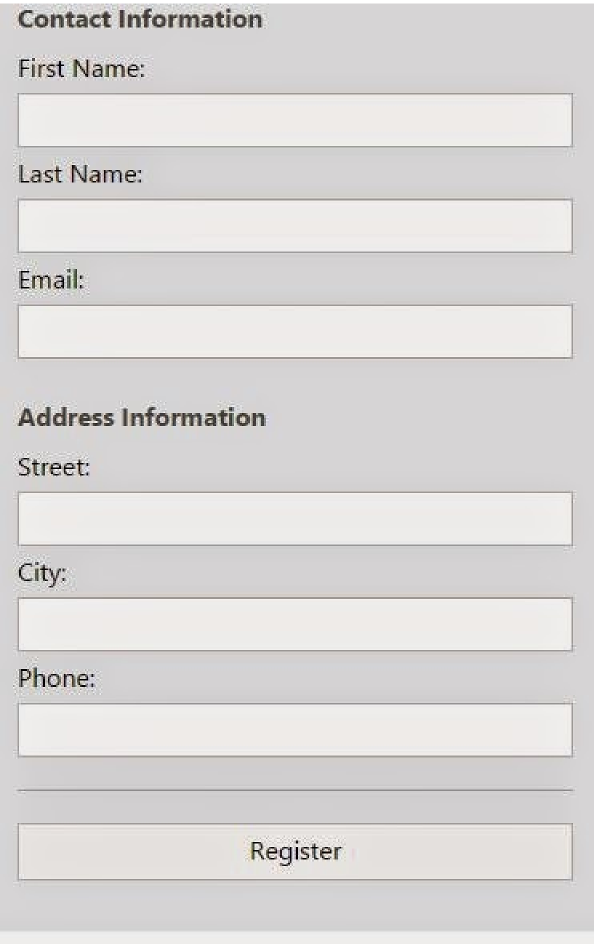
Pageviews by Browsers

Entry
Internet Explorer
Firefox
Chrome
Safari
Mobile
CriOS
Opera
BingPreview

Test Website Access

Well, typically the website you're producing is kept tightly under wraps until it's launched. Don't forget to have a discussion about that with your web team. Do you have a VPN[19] you can set up on devices to access your test environment? You're going to need some form of solution to get your device to see your test pages.

Can't we just get around needing mobile devices, and just use the browser in small mode?



[19] http://en.wikipedia.org/wiki/Virtual_private_network

If you make your browser 480 x 800 - isn't that the same as using a browser?

It's to be fair a good first step, but as you'll see below, some of the problems come from the operating system. Android and iOS have special build in ways to handle certain items like drop down and pop-ups which mean they behave slightly unexpectedly.

So I'm set up - what next?

Okay, so someone approved your devices, you have an accessible test area and you're ready to go … and?

So what exactly are you going to test? What really helps is to come up with a way to summarise your application. To cross browser and cross device test you need to repeat the same elements of testing over and again for every browser and device.

Do you have any clear idea what those elements are for your system?

The following is a guide to my approach … and remember I looked at some of this in the previous section on browsers.

Page checking

Create a map of every page. Confirm that,

- can every field can be selected and data entered?
- can every check box selected/unselected?
- can every drop down box selected?
- can every button be selected?

- can you select between tabbed pages?
- check for pop up confirmation and error messages

You are basically looking here for something not working, a button missing etc which would prevent you from being able to use a page!

Functional Checking

What are the main basic functions of the website, I need to make a list of them, and repeat on several browsers and devices.

Here's some examples *(discussed previously)*,

- Registration
- Login
- Change my account details

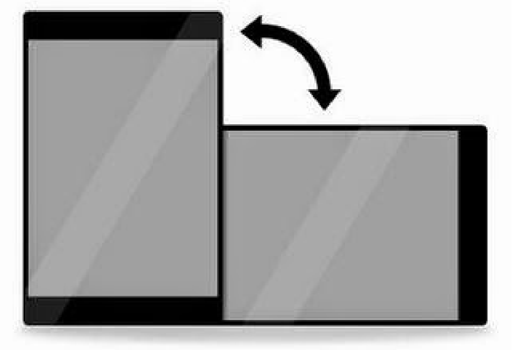
Generally the tests per browser/device don't have to be exhaustive - you are repeating a few examples across multiple browsers after all. But ideally should include a success and a failure *(you always want to check an error message is displayed)*.

Being able to create a summary overview for testing of your website, is something I hope to explore in the future - so watch this space.

Common Gotcha's

These are the areas I've commonly found problems with on responsive design. You'll notice that some of these can be countered by good, up-front design … so if as a tester you find yourself in a design meeting, go in forearmed …

Landscape to portrait to landscape

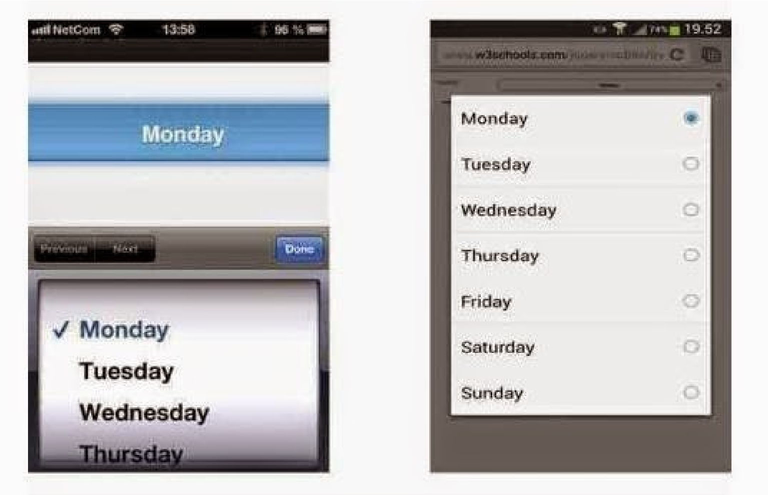


A simple test - part fill your page in, turn it on it's side. Do you lose your data? Does any page element vanish?

Turn it back to it's original orientation, and check again. Sometimes the changed orientation causes a page refresh, and things go missing!

Drop downs

The mobile device overrides how drop downs are handled in browsers …



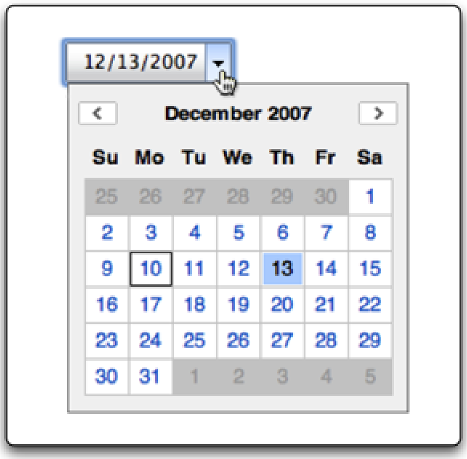
On the left is the iOS carousel, and on the right the Android selection list.

The problem though occurs if you have long selectable items on your drop downs, For example, consider a list of security questions,

- What was the name of your first girlfriend/boyfriend?
- What was the name of your favourite film?
- What was the name of your junior school?

All these will truncate the questions, so all the user will see is *“What was the …“*. There’s really no solution for this, but rethinking the approach/redesign.

Pop ups



12/13/2007

December 2007

Su	Mo	Tu	We	Th	Fr	Sa
25	26	27	28	29	30	1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31	1	2	3	4	5

I've found any kind of required pop ups from a browser can be a little troublesome on mobile devices (they often just go AWOL).

This can include,

- Are you sure you want to continue?
- Read these terms and conditions.
- Select a date from this calendar

Tread carefully - and always check these.

The catch with responsive design

Yeah - there was bound to be one, wasn't there? This approach significantly reduces risk of problems in responsive websites, but it's no guarantee. Indeed some mobile phones take a system like Android and tailor particular features, so it's hard and expensive to exhaustively test upfront. This risk has to be known to both yourself and your client.

Furthermore, because responsive design uses newer features of HTML, it means that much older browsers and devices really don’t handle the page very well. You might want to consider that on old, out of scope browsers/devices that it at least fails gracefully with a “your browser does not support this website” error message.

Some common sense ticket items

Finally some common sense things to consider before we wrap up …

Health and safety

Some testers are really keen to get into mobile testing. I actually really find it a pain. The screens are much smaller, it requires data entry just with your thumbs, you tend to hunch over the device.

This is a recipe for repetitive strain injury. Try and mix up mobile testing with browser testing, and make sure you’re taking regular breaks.

Keep them safe and keep them secret

Make sure they’re locked away when not in use, though have a couple of keys to where they’re kept with your team. You just don’t want them walking away. Lock and count them in every night.

You drop it you bought it?

It’s inevitable that one is going to get dropped. What then? Look into getting them added and listed to your building insurance. If you can’t make it clear what will happen should any damage occur.

Chapter 10 – The world wide test community

And there we have it – if you've come this far, then you've managed to flex your tester muscles and apply some of the core principles we apply daily as testers. Well done.

So what next?

I want to close the book by looking through some resources which are available online for you to engage with. Some are sites or magazines, others are key figures in software testing who might be active either through book writing, blogging or on Twitter.

Twitter has become a really important place for us as testers – it's where a lot of testers go to share articles, ask for help, or discuss ideas. If you haven't already, I advise you to create a Twitter account, and select to follow some of the people I've picked out. *[Well I advise you follow them all]*

It's a public place, and some of the discussions can look like arguments, especially with people like myself who are context driven in their approach to testing. Generally people share their experience, put forward ideas, then because we're testers we look for holes in that idea, and sometimes come up with different ideas. That's how we evolve our understanding of testing.

If you ever get the opportunity to attend a peer conference, you will see that a similar approach to discussing key ideas.

Context driven testers believe there is no single foolproof way to approach testing – every project has different business drivers, technology and people. So a good tester knows they need to tweak what they do, and sometimes take a giant leap outside of what they know.

When I look at the range of topics people discuss and write about within testing, I see there being two key areas,

- **Test Mechanics** – the core of this book is about the mechanics of testing, indeed "how to test". How you apply testing principles, strategies for using particular tools etc.
- **Test Relationships** – as you progress in seniority this field becomes more important, although it always has a level of importance. How you deal with other testers, developers, managers and especially how you work towards a common goal, and help develop their understanding of testing factors and issue.

It's taken me a while to compile this list, which I've tried to keep constrained to just testers. Even so, I just had a flurry of "don't forget X" just when I thought I was finished. Apologies if I've missed someone, and do get in touch if there's someone I need to add! [Remember what I said about human fallibility right at the start?]

Test Resources

Ministry Of Testing

Website: https://www.ministryoftesting.com/

Twitter accounts: @ministryoftesting, @rosiesherry, @sjpknight

This is a site which started as just a forum about 6 years ago, and has grown and evolved under the leadership of Rosie Sherry. You can sign up for a free account, and you should have access to a good amount of material, including articles and videos.

I previously helped out as a reviewer on the Ministry Of Test's Testing Planet, and worked closely with editor Simon Knight, who I consider an all around "good sort".

Testing Circus

Website: http://www.testingcircus.com/

Twitter accounts: @TestingCircus, @ajoysingha

A lot of material from this book was originally printed in Testing Circus, a monthly magazine for testers. Well worth reading, with a nice supporting website of past articles.

Sticky Minds

Website: http://www.stickyminds.com/

Twitter accounts: @StickyMinds

Another site which has evolved out of being mainly just a forum several years ago. It has weekly articles which draw on a range of experience. And it costs nothing to sign up.

Teatime With Testers

Website: http://www.teatimewithtesters.com/

Twitter accounts: @TtimewidTesters

Another magazine I've written for. So of course I'll recommend them.

Testing Folk

In no particular order, this is a list of people who I find interesting to either read up about, or engage with about testing. I'm focusing particularly on people who're very active at the moment or who have had a big impact.

Mike Talks

Website: http://testsheepnz.blogspot.co.nz/

Twitter account: @TestSheepNZ

Starting with myself. Although this book is about the ideas of test mechanics, I mainly write and explore about test relationship ideas such as teamwork on my blog.

My preferred method of being contacted is to drop me a line on Twitter, where we can explore the topic together as time allows.

Bernice Niel Ruhland

Website: http://thetestersedge.com/

Twitter account: @bruhland2000

Bernice and I talked a lot about the themes of this book back in 2012, but we never had time to do them justice. She has written a several great themed series for Testing Circus.

James Bach

Website: http://www.satisfice.com/

Twitter account: @jamesmarcusbach

James is a leader within the context driven test community, and the originator of the session based test management techniques which drove our chapter on exploratory testing. His writing is very strong about teaching *test mechanics*.

He is one of the authors of Lessons Learned In Software Testing: A Context Driven Approach, one of the most influential books on software testing ever written. This book explores key problems and areas that testers will find themselves exposed to throughout their career.

He also tours the world with his highly successful Rapid Software Testing course – however to get the most of this course, it really helped to have had a couple of years actually within testing so that you can bring and apply your experience in the class.

Elisabeth Hendrickson

Website: http://testobsessed.com/

Twitter account: @ testobsessed

Although Elisabeth isn't quite as active online as she used to, she has a very fine back catalogue of pieces to read through. She is the author of the Heuristic Cheat Sheet we've used throughout chapter 4, and has her own book Explore It on exploratory testing.

I really enjoy Elisabeth's writing, which is often quirky and fun. I only noticed recently what a huge influence it had on me, because it taught me "writing about testing doesn't have to be academic and boring". There are a few videos of her on YouTube, where her passion and energy for the topic is evident.

When Googling her, if you get the Mad Men actress, please try typing her name more carefully! [It's happened before when I've recommended her]

Lisa Crispin & Janet Gregory

Websites: http://lisacrispin.com/ & http://janetgregory.ca/

Twitter accounts: @lisacrispin, @janetgregoryca

In this book I’ve steered clear of software lifecycle methodologies. Mainly because every project has their own homebrew lifecycle due to the individual limitations and context of your project.

However, I can recommend these co-authors for their work in the field of agile testing. Back when agile was starting to breakthrough as a dependable method of delivering software, there was also a lot confused whispering going on about “agile teams don’t need testers”.

Their book Agile Testing: A Practical Guide For Testers And Agile Teams really looked to get rid of the confusion. To teach testers used to waterfall what agile was. And to look at techniques testers could use to work with the agile format, rather than against it *(as many of us felt at the time).*

Their book was a must read a few years ago for people trying to take the step into agile *(I was one of those people)*, and was an introduction to some key ideas such as the appropriate use of automated checking to support manual testing, and the application of mind maps and exploratory testing.

Like myself, Lisa and Janets writing is very focused on *test relationships*, building a place for testing within a team dynamic, although their book includes introduction to key *test mechanic* ideas.

I worked as a reviewer on their follow up book More Agile Testing.

Michael Bolton

Website: http://www.developsense.com/

Twitter account: @michaelbolton

Michael is another context driven tester, and often to be found actively on Twitter discussing test ideas. His blog also contains a lot of very good source material.

And no, he is not the "sitting on the dock of a bay" Michael Bolton. And as someone who has a joke surname, that joke wears real thin, real fast.

Jerry Weinberg

Website: http://www.geraldmweinberg.com/Site/Home.html

Twitter accounts: @JerryWeinberg

Jerry has written extensively about software, leadership and consulting. For me, two of his more influential books are The Psychology Of Computer Programming and Perfect Software And Other Illusions About Testing. But he's written so much, I doubt you could get two people to agree on their favourites!

I primarily enjoy his writing on *test relationship* topics.

Guna Petrova

Twitter account: @alt_lv

As my student, and close friend, Guna gets a special mention. She's not got a book, or a huge backlog of blogs, but she's working on it.

Guna a few years ago was where you are now. She's applied incredible energy into learning about software testing, and also has shared that passion doing public speaking about why she enjoys the challenges of the career.

She has managed to get attendance to events like TestBash and Nordic Test Days through volunteering. As such, I feel she's a rather stellar role model for young testers, hence how she earns a place in this list.

Kiwi testers

This is a listing of testers who live and work in New Zealand with me, together with their twitter handle. I've met up in person with all of these people. Many areas around the world have some form of local meetup which it's well worth going along to.

- Sarah Burgess, @KiwiDudette
- Rachel Carson, @akiwitester
- Katrina Clokie, @katrina_tester, *Testing Trapeze editor*
- Sean Cresswell, @seancresswell
- Michele Cross, @MichelePlayfair
- Dave, @nzkarit, *Our local red-shirt security officer of testing*
- Kim Engel, @kengel100
- Oliver Erlewein, @Oliver_NZ, *Organiser of the annual Kiwi Software Testing Workshop*
- Aaron Hodder, @AWGHodder, *Mr Mind Maps*
- Adam Howard, @adammhoward
- Moss Nye, @mossnz
- Chris Priest, @cbpriest78
- Joshua Raine, @raine_check
- Rich Robinson, @richrichnz
- Chris Saunders, @testRUNnz

Passionate testers

Okay, this is not to say people who aren't on this list aren't passionate. But these are people I've noted for their talks about testing, who don't quite fit on one of the other lists.

- Ajay Balamurugadas, @ajay14f
- Fiona Charles, @FionaCCharles

- Terry Charles, @booksrg8
- Anne-Marie Charrett, @charrett
- Erik Davis, @erikid
- David Greenlees, @DMGreenlees
- Parimala Hariprasad, @CuriousTester
- Jean Ann Harrison, @JA_Harrison
- Matt Heusser, @mheusser
- Stephen Janaway, @stephenjanaway, *Mobile guru*
- Helena Jeret-Mäe, @HelenaJ_M
- Karen Johnson, @karennjohnson
- Trish Khoo, @hogfish
- Keith Klain, @KeithKlain
- Alessandra Moreira, @testchick
- Claire Moss, @aclairefication
- Nicola Owen, @NicolaO55
- Erik Petersen, @erik_petersen
- Patrick Prill, @TestPappy
- Maaret Pyhäjärvi, @maaretp
- Anna Royzman, @QA_nna
- Huib Schoots, @huibschoots
- Ben Simo, @QualityFrog

The kids at the back of the bus

Without doubt, this is my favourite bracket of contacts. If you look through who I'm following, you'll see there is a lot outside of just testing. I follow managers and agile coaches and historians.

This group of people are testers I can talk to about testing, and almost anything else under the sun. *We're people who are having fun!*

- Dan Billing, @TheTestDoctor, *Sci-Fi aficionado*

- Doug Buck, @Stormgod, *Games Workshop fan*
- Lanette Creamer, @lanettecream *Humour & cats*
- Gabrielle, @theGuttedTester, *Space explorer & Tinkerbell advocate*
- Tim Hall, @Kalyr, *Music & railways "green for danger"*