There's a game in an arcade, on a pier near here. You face a board full of holes, and you get ready with a big, soft mallet. The game starts - a mole pokes its head out of a hole. You whack it, and it goes away. Another mole - two moles - you whack both of them. One mole - whack. Long pause. Three moles - whackwhackwhack. You get points for whacking moles. Sometimes, you won't notice a mole, sometimes more moles appear than you can easily deal with. Moles left standing go away after a while - you get no points for those moles.

Welcome to my world of exploratory testing. As I explore software, interesting things pop up.

Is it a bug? Whack.

Another interesting thing. Not sure if it's a bug - is there another test? Whackwhack.

Long pause.

Interesting thing! Whackwhackwhack . . . and it's not as interesting any more; I can move on.

Sometimes, I see lots of interesting things. Sometimes, I miss interesting things. Unlike moles, I can come back to the interesting things later, so sometimes I just write them down. All well and good, but how can I tell what's interesting? I use my judgement.

- = = -

Judgement's a tricky fellow to pin down. In the real world, judgement is about decisions. Important decisions; deciding whether something is good or bad - and by how much. Deciding if something needs to be done, right now, or if it can be left for a while. Judgement should be impartial, consistent, defendable - which feels like testing to me. Testing and judgement are entwined.

This perhaps explains why I get so irritated (in a positive, polite, but above all *internal* way if you're a client of mine) when some fool of a tester tells me that they can't even start to tell me where the problems might be without a complete specification. Or if I'm showered with indiscriminate defect reports from some spec-less bug-jockey who simply decides that everything they don't personally like is and has to be a fundamental flaw in the code, the technology, the design, even unto the very soul of the poor forgotten programmer who first laid calloused hand to Qwerty. Where's their [expletive deleted] judgement? Where's my mallet?!

- = = -

Enough. Nothing's perfect. They're not bad people. Calm. Relax. Breathe.

Let me tell you what I use to judge bugs. Then, I'll tell you how important judgement is to finding good bugs as easily as possible.

I have a framework to help me judge bugs. Not a scale - it is, it might be, it isn't - but a *framework*, a set of ways of thinking about what I find. Not an exclusive classification, either; things can be interesting for more than one reason. All the better. There's no first or second pass, no process; I try to keep the whole framework in mind. It isn't hard as I've made it sound, so let's give it a go.

When I explore, I think about **inconsistencies**, **absences**, **extras**. Does a button work in a different way there from here? Why has the system forgotten my data? Does it look like there's another way of doing this?

I also try to identify what I might be comparing against. Something **internal** to the system I'm working with, perhaps - another screen, an older version, a different account. If not internal, then something **external**. Either something **specific**, like a dictionary, a timetable, (heaven forbid) a specification - or something **cultural**, like a way of working, a general principle, an expectation.

Some examples would help, and you'll find some at the bottom of the page. However, it's important to understand that I'm *not* looking for these examples, nor am I necessarily thinking of what to look for before looking for it. Instead, I'm exploring the *system*, looking at the *system*. I'm thinking, at the back of my mind, of my framework. My subconscious does a better job of noticing interesting things than my conscious (which is generally better at concentrating on them). The framework sits between the two. As I explore the system, interesting things appear - this framework helps me notice them more easily, and understand something perhaps about why they're interesting.

- = = -

Some fundamental process of the mind allows me to separate the interesting from the dull, the important from the noisy, the unusual from the expected. The shape from the shadows - the tiger, indeed, from the jungle. Some things I miss - and some things I do notice aren't really there. If I can judge whether something is interesting, or is not - impartially, consistently, and in a way that I'm sure I can defend my position - my choices are more clear, my decisions more supported, my exploration more focussed.

It seems, too, that I get better with practice - the more I use the framework, the better I am at noticing the subtler things, the better I am at filtering out the false positives. My subconscious, eager to please, responds to feedback. When I learnt to drive, every corner, every tree, every cyclist occupied my all-too-thinly-stretched attention. Now, something tells me when the pedestrian hasn't seen me, when that driver's turning without using his indicators. My judgement has (we hope) improved. Exploration is the same. As ever, there's a virtuous circle here, if you can just get on the ride: Good judgement is *consistent*. Consistent feedback helps you *learn*. What you're learning is to be a better *judge*.

- = = -

Consciously using your noggin is good for the soul - and it makes you a better tester. But there's more. Judgement lets one *refine* ones exploration. Let's look at some situations where judgement plays an integral part in test design.

First, something trivial. A button is labelled 'stap'. It's interesting, because we don't recognise the term. That's obvious to us because we're English speakers, the application's in English, and *stap* isn't in the dictionary. The label is inconsistent with an external specification. Do we even need to check? Probably not. It's a buggety bug. Let's log it and keep on going.

A more subtle one now. Let's imagine the button is labelled 'step'. It's not as interesting, doesn't stick out as much, because step is in our personal lexicon. We've noticed it, but we need more tests to decide if it's a bug. Press the button. It appears that the button *stop*s a clock. A spelling mistake, just like the previous example? Press it again - the clock restarts. Hmm. Perhaps *step* is an appropriate label after all.

We didn't have enough information to judge, so we kept going. Where next? Internally? Is there another button with the same function - what's its label? Externally - a real spec? an cultural inconsistency like an unwritten rule, a customer expectation? Off we go.

- = = -

Working with a framework for judgement drives us to test, and test again, until we have a better idea of what we're dealing with. A framework for judgement is important because it leads along the path from less to more certainty, indicating alternate understandings along the way.

However, recognition and judgement don't always go together. Internal inconsistency is easy to spot - and hard to judge. A red button here - a green button there: You can be sure that one of them is wrong. Perhaps they both are. If there are no other clues, that's all you've got. Log the difference, and move on. Testers don't change lightbulbs, they just tell you the room's gone dark.

- = = -

External inconsistency is tricker to spot - you need to have a clue what's outside the system to compare with. Without an understanding of what's outside, you can't hope to spot an inconsistency. If you've read every scrap of documentation, you might find it easier to recognise problems against the specification. If you've got plenty of customer experience, you might find it easier to recognise unwritten problems.

However, if the external sources are impeccable, you've found a bug. Not all

sources are. Specifications are often wrong - you'll have to judge. Sometimes you'll judge against a standard. If compliance is mandatory, and the standard is wrong, you'll just have to bend your application to suit. Of course, if you're dealing with user expectations, the customer is always right.

- = = -

It's not just about inconsistency. Perhaps something is interesting because it is missing, or has been added. If you restrict yourself to thinking of software components, the difference between these two is often one of versioning - 1.0 does, 1.1 doesn't. If you think of external sources, ideas of what might be missing, or what might be included, become much more interesting.

This is my simple framework. It doesn't say much about time, or about multiples. It doesn't help me judge against exploitations, technologies, styles of use, attacks. Since I notice these things, and can judge one to be more worthy of my attention than others, I assume that I have more frameworks. I've not yet uncovered what they might be. If this is familiar to you, perhaps you have your own frameworks. Perhaps you already have an idea what they are - I look forward to hearing about them from you. In the meantime, though, I'll leave you with these examples, and let you get on with your thinking.

	Internal	External: Specific	External: Cultural
Inconsistency	The address has three lines here - but four over there Usually that beast explodes when I shoot it with the railgun Seems to have got faster just now	The spec says I should be prevented from doing this This file won't make sense at the interface London is on the *east* coast The timetable says no trains after 12:30	People want to see their total on the *first* page This takes too long to start up What do you mean, 'state' is a required field? Midnight is *not* downtime for this cab company
Absence	Last version, I could address an email from here Where'd the button go?	I *should* be able to add a column There is no character validation on this field I can't see the two-player mode	This field is too small for my email address Where's the privacy policy? These people have no manners
	Where'd that new button come from ?	There's nothing	Isn't that my

Extra	Three places I can change the title - do they all change the filename? If I had two logins, how would it decide which to listen to?	There's nothing in the spec about this easter egg Does the spec actually include paste-via-keys? Looks like I can insert a picture after all	Isn't that my password? But I don't want to 'show everybody that I am idle' Puce? I'm going to spend all day looking at puce?