Combinatorial Software Test Design

(Beyond Pairwise Testing)

Presented at STPCon

October, 2010

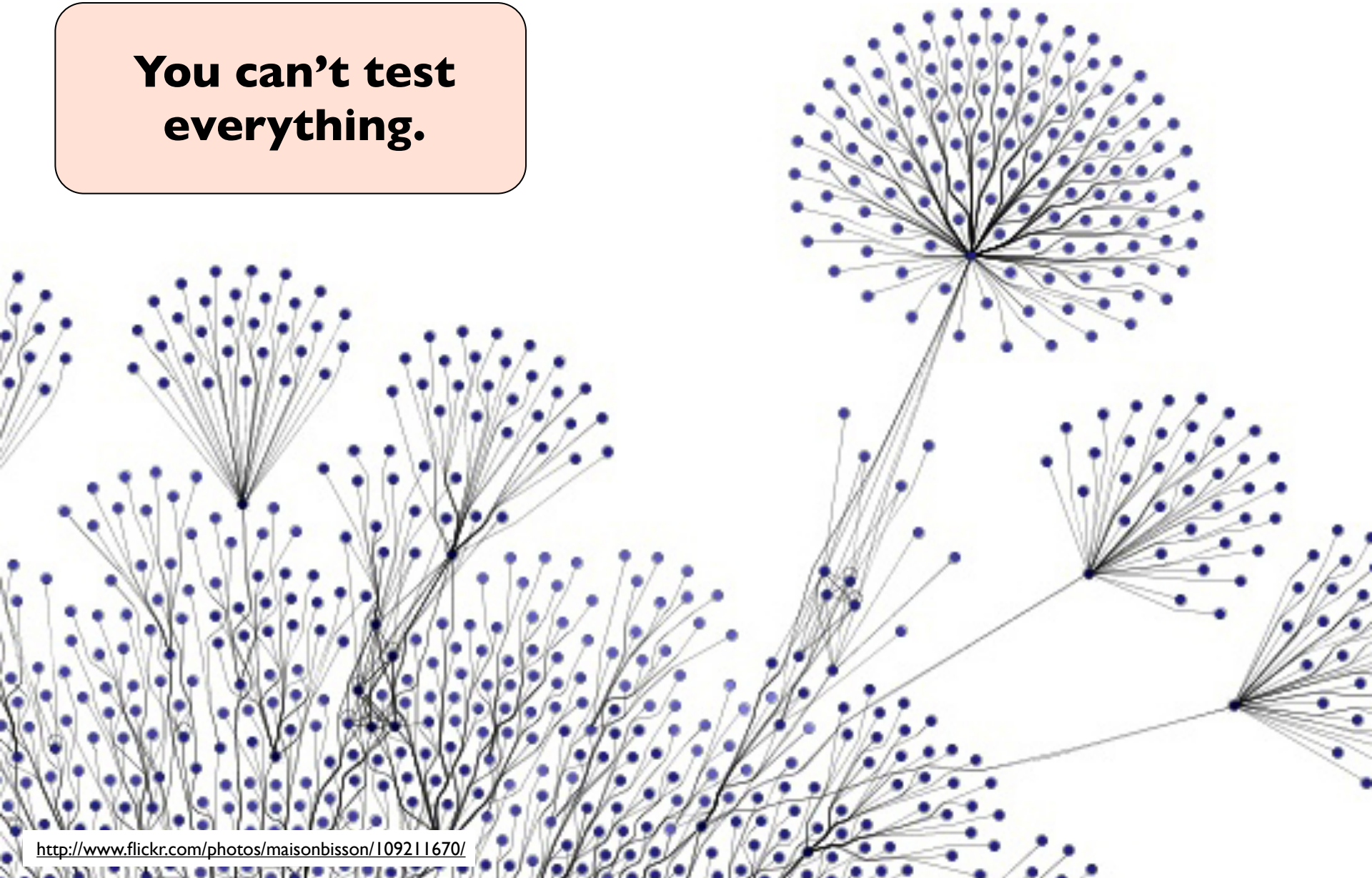
By Justin Hunter
CEO of Hexawise

Cover slide photo credit: "Mike" Michael L. Baird, flickr.bairdphotos.com
http://www.flickr.com/photos/mikebaird/2127310513

PROBLEM

The Problem

You can't test everything.



http://www.flickr.com/photos/maisonbisson/109211670/

The Problem

http://www.flickr.com/photos/madabandon/344158061/

The Problem

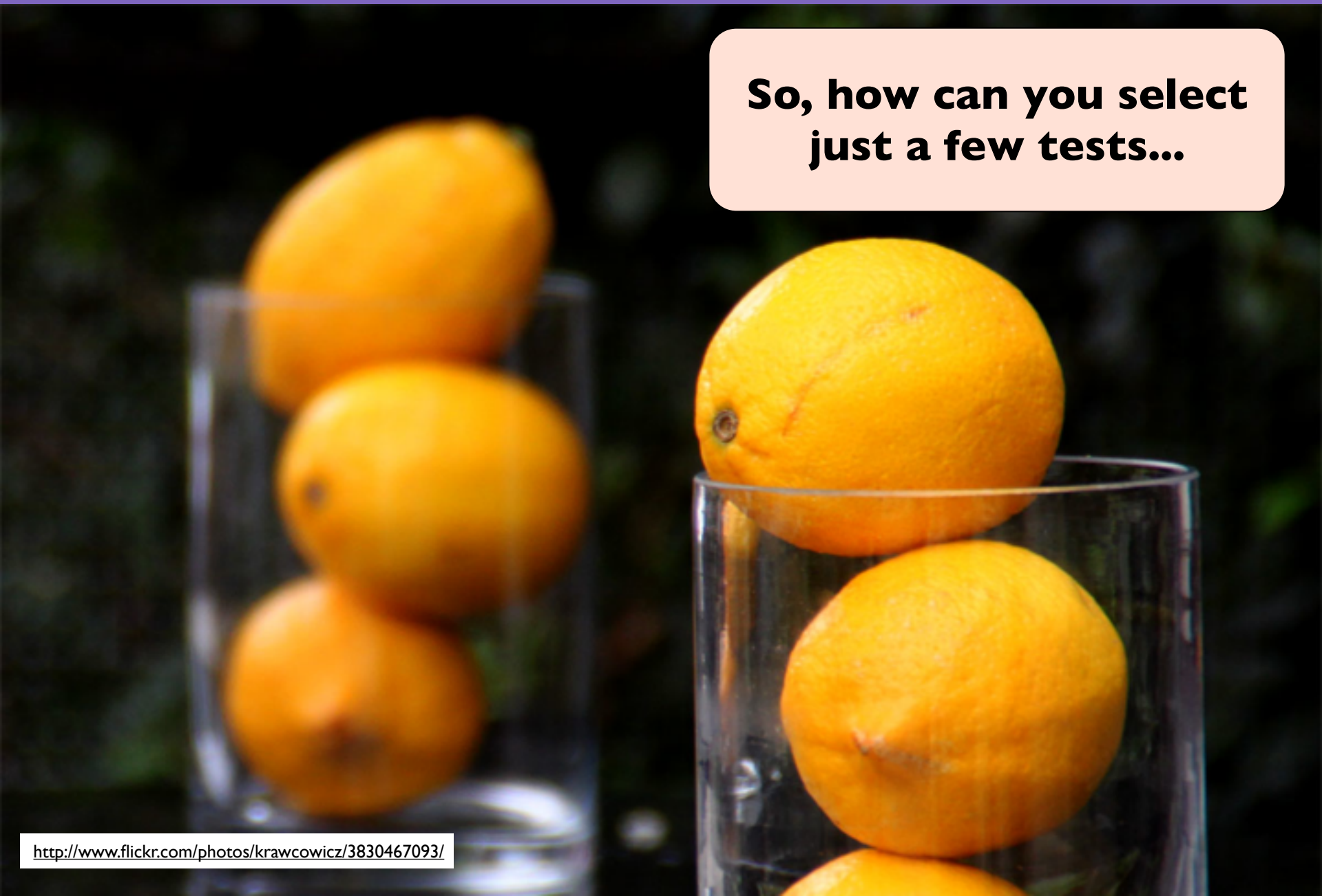
http://www.flickr.com/photos/maisonbisson/110867288/sizes/l/in/photostream/

The Problem

... discouraging, even.

http://www.flickr.com/photos/cheo70/2702682262/

The Problem


So, how can you select just a few tests...
http://www.flickr.com/photos/krawcowicz/3830467093/

The Problem

... and still achieve adequate coverage?

http://www.flickr.com/photos/editor/4861496460/

The Problem

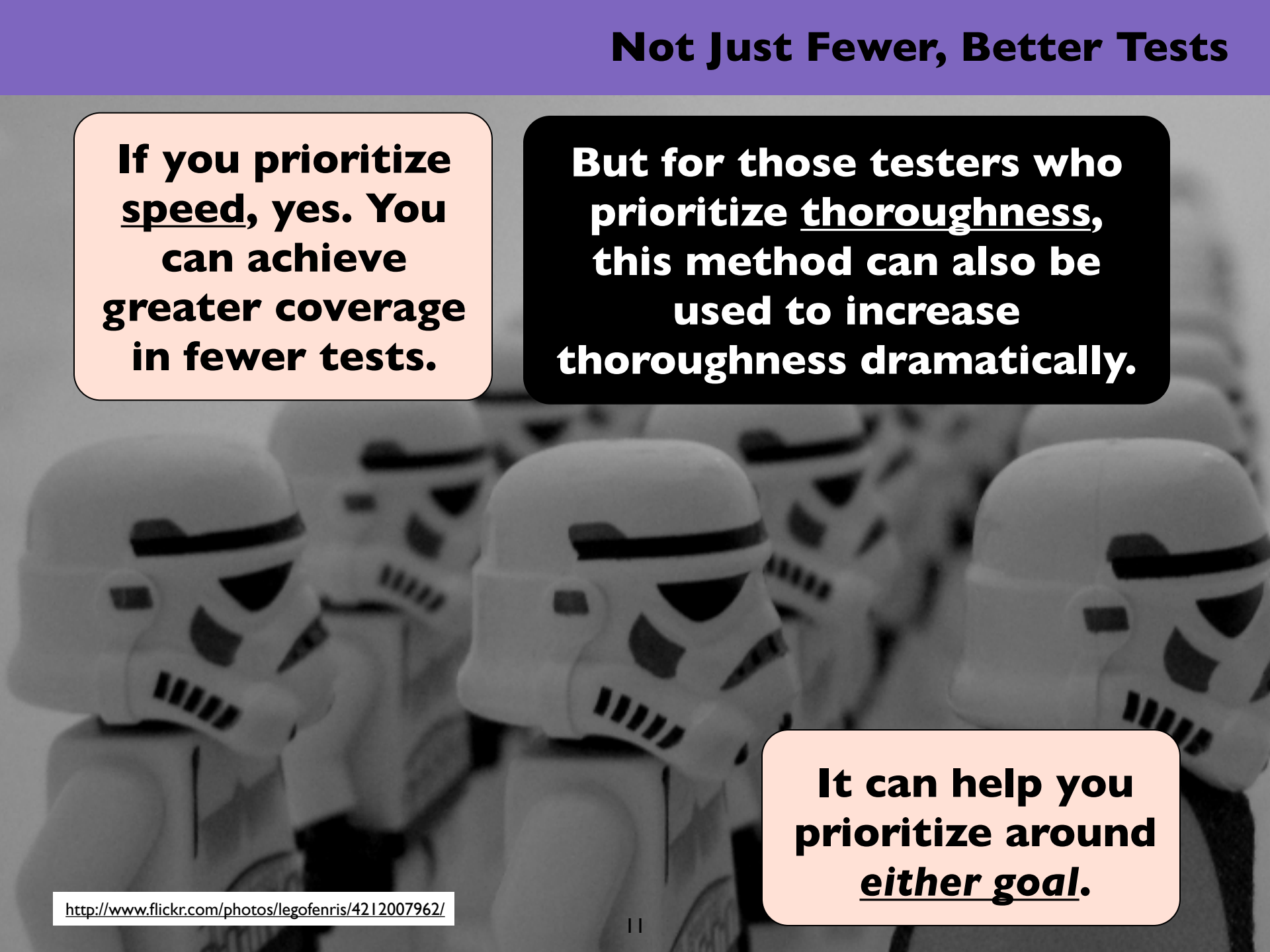
Wait. So this is a test design method to achieve greater coverage in fewer tests. Right?

http://www.flickr.com/photos/johncarleton/4890239248

Not Just Fewer, Better Tests

If you prioritize speed, yes. You can achieve greater coverage in fewer tests.

But for those testers who prioritize thoroughness, this method can also be used to increase thoroughness dramatically.



It can help you prioritize around *either goal*.

http://www.flickr.com/photos/legofenris/4212007962/

Not Just Fewer, Better Tests

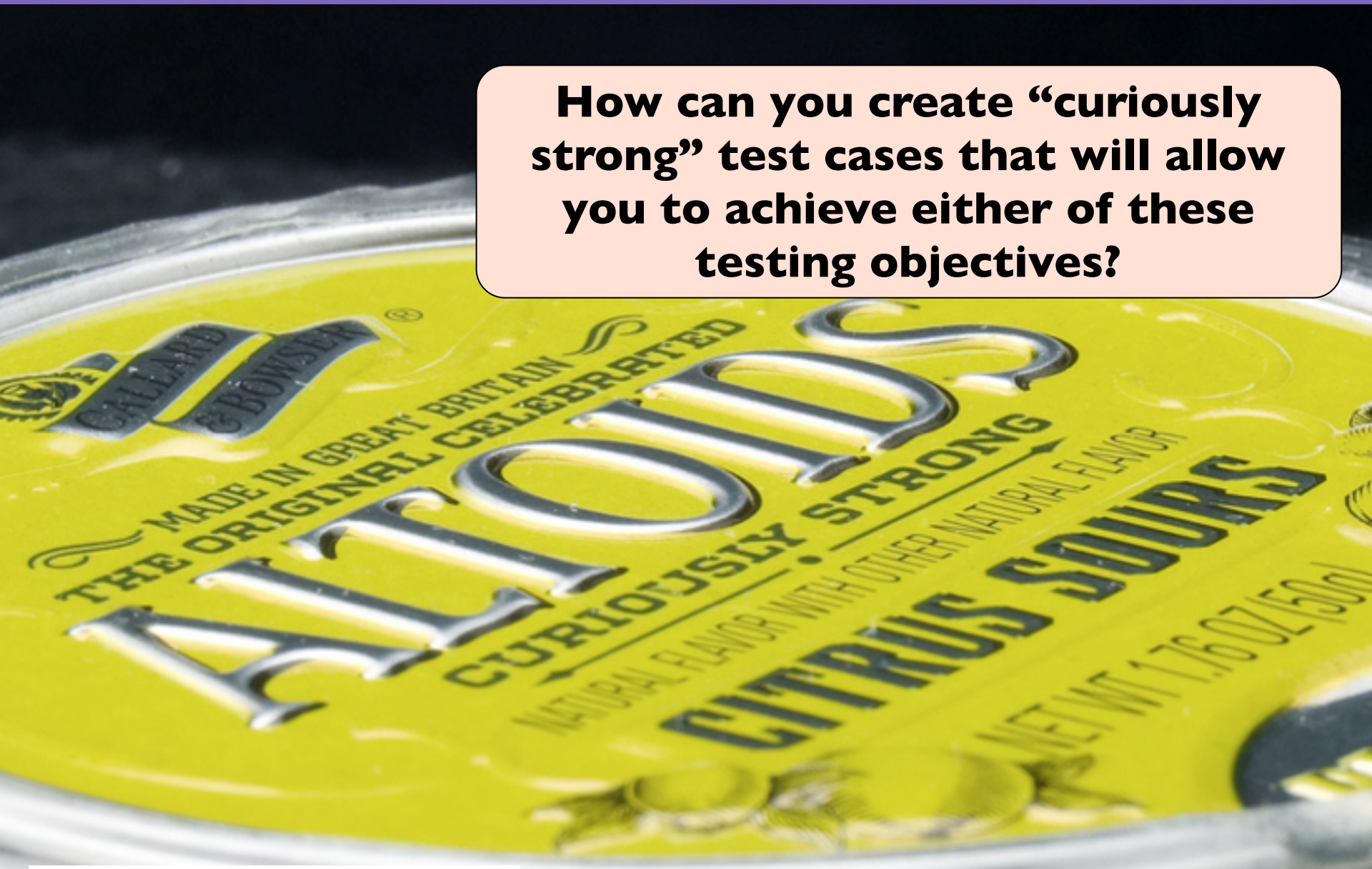
The choice is yours to make.

http://www.flickr.com/photos/thomashawk/2402194148

http://www.flickr.com/photos/thomashawk/2402190314

Not Just Fewer, Better Tests

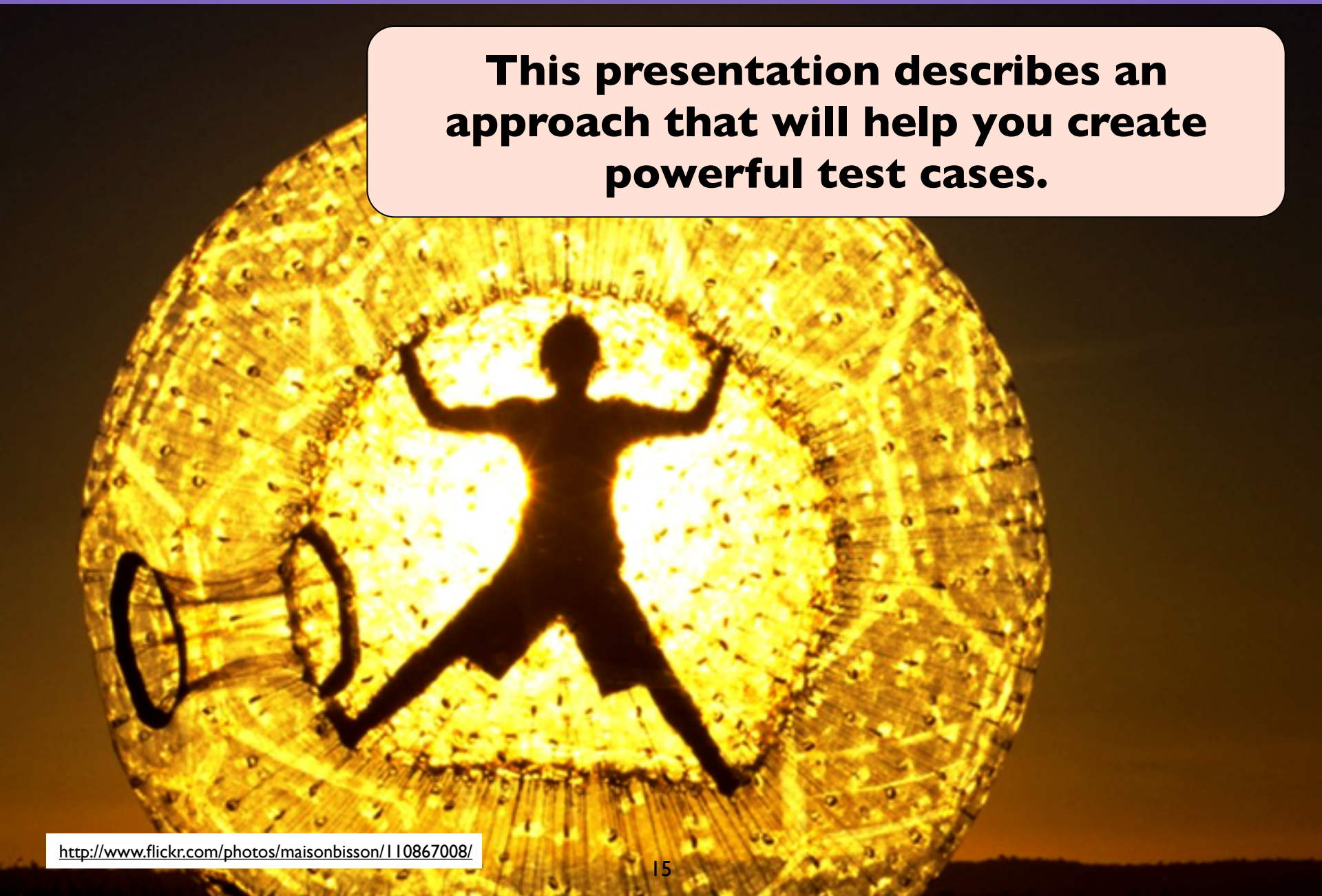
How can you create "curiously strong" test cases that will allow you to achieve either of these testing objectives?



http://www.flickr.com/photos/photobunny_earl/2309612944/

SOLUTION

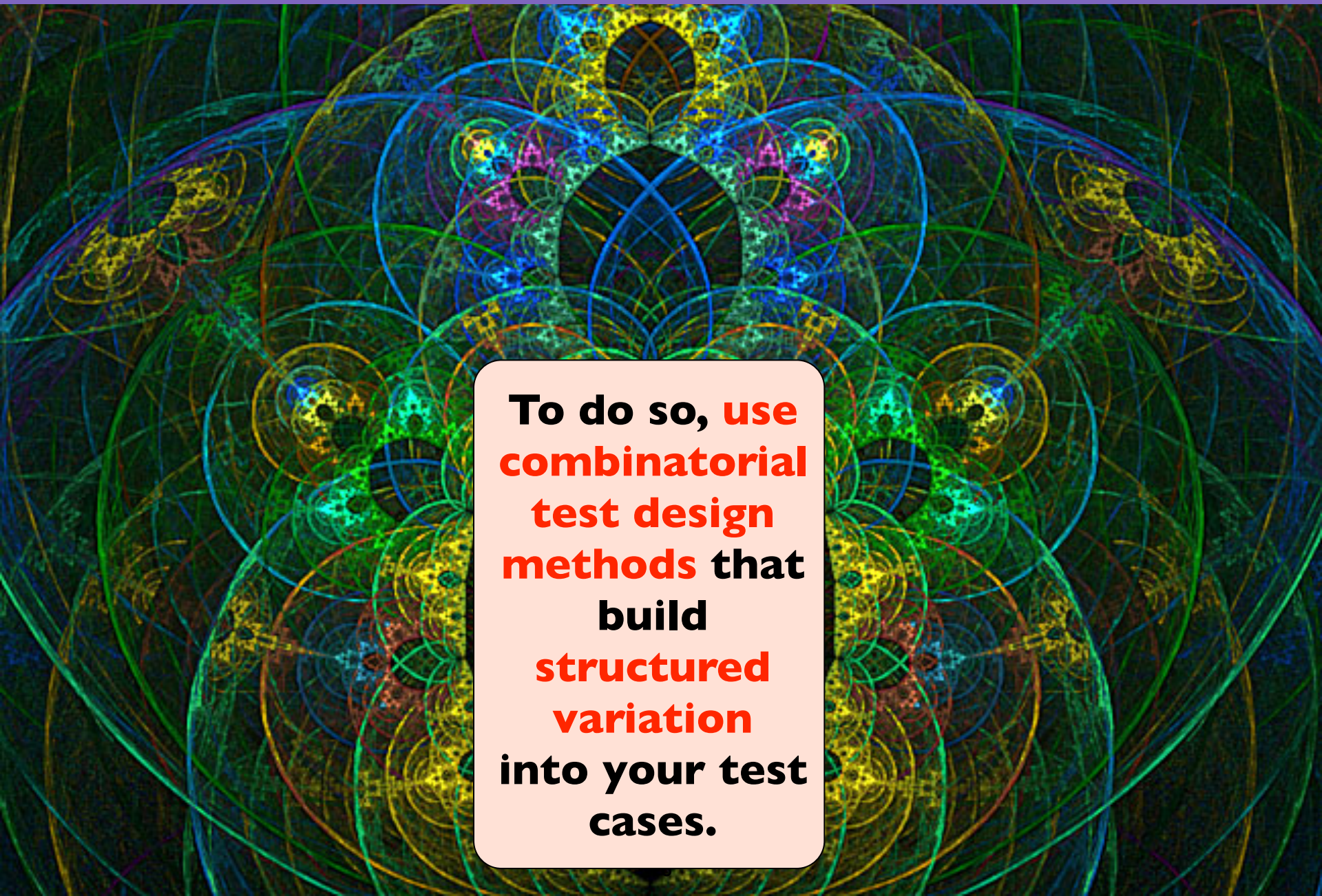
The Solution



This presentation describes an approach that will help you create powerful test cases.

http://www.flickr.com/photos/maisonbisson/110867008/

Structured Variation



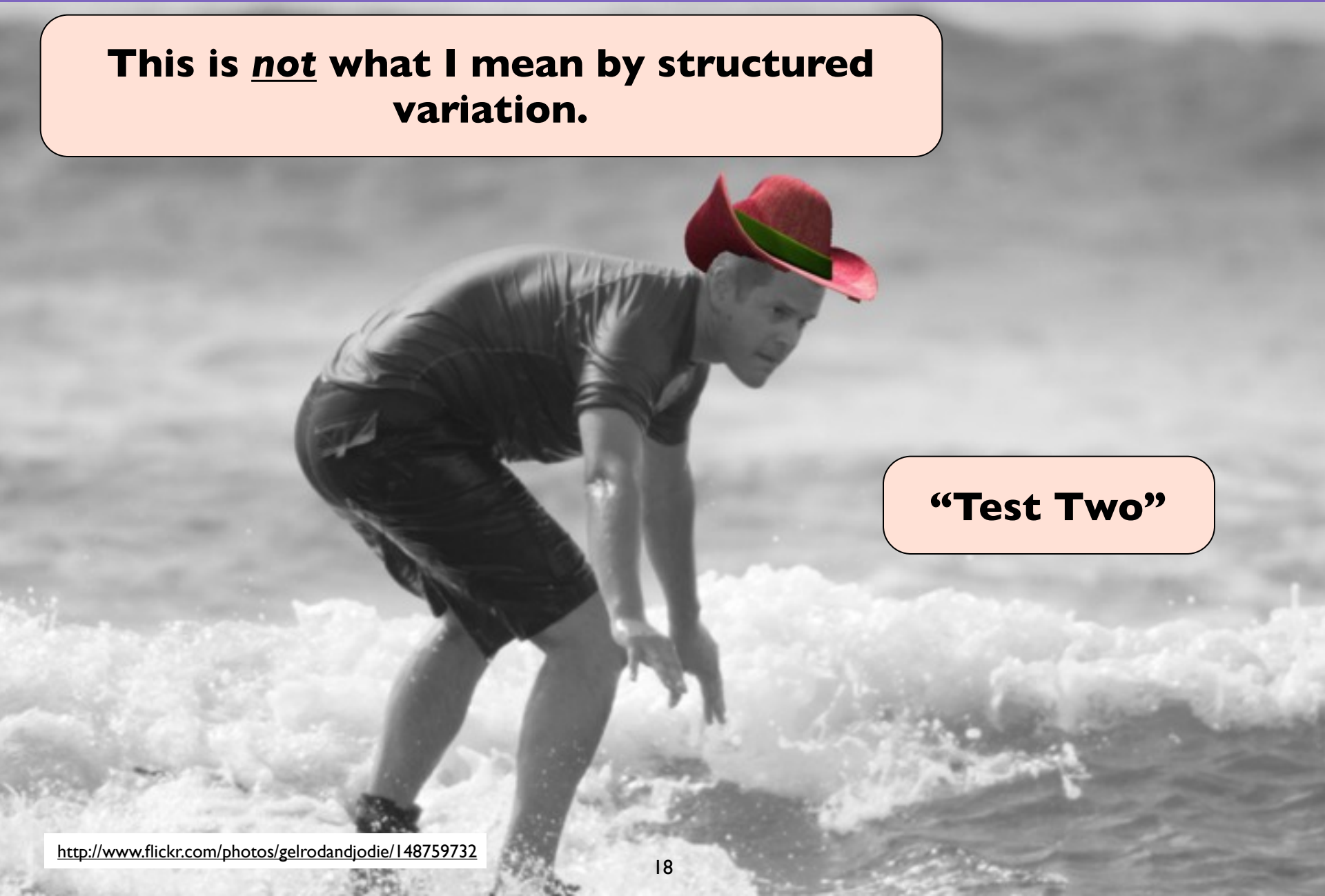
To do so, use combinatorial test design methods that build structured variation into your test cases.

Structured Variation?

http://www.flickr.com/photos/gelrodandjodie/148759732

Structured Variation?

This is *not* what I mean by structured variation.

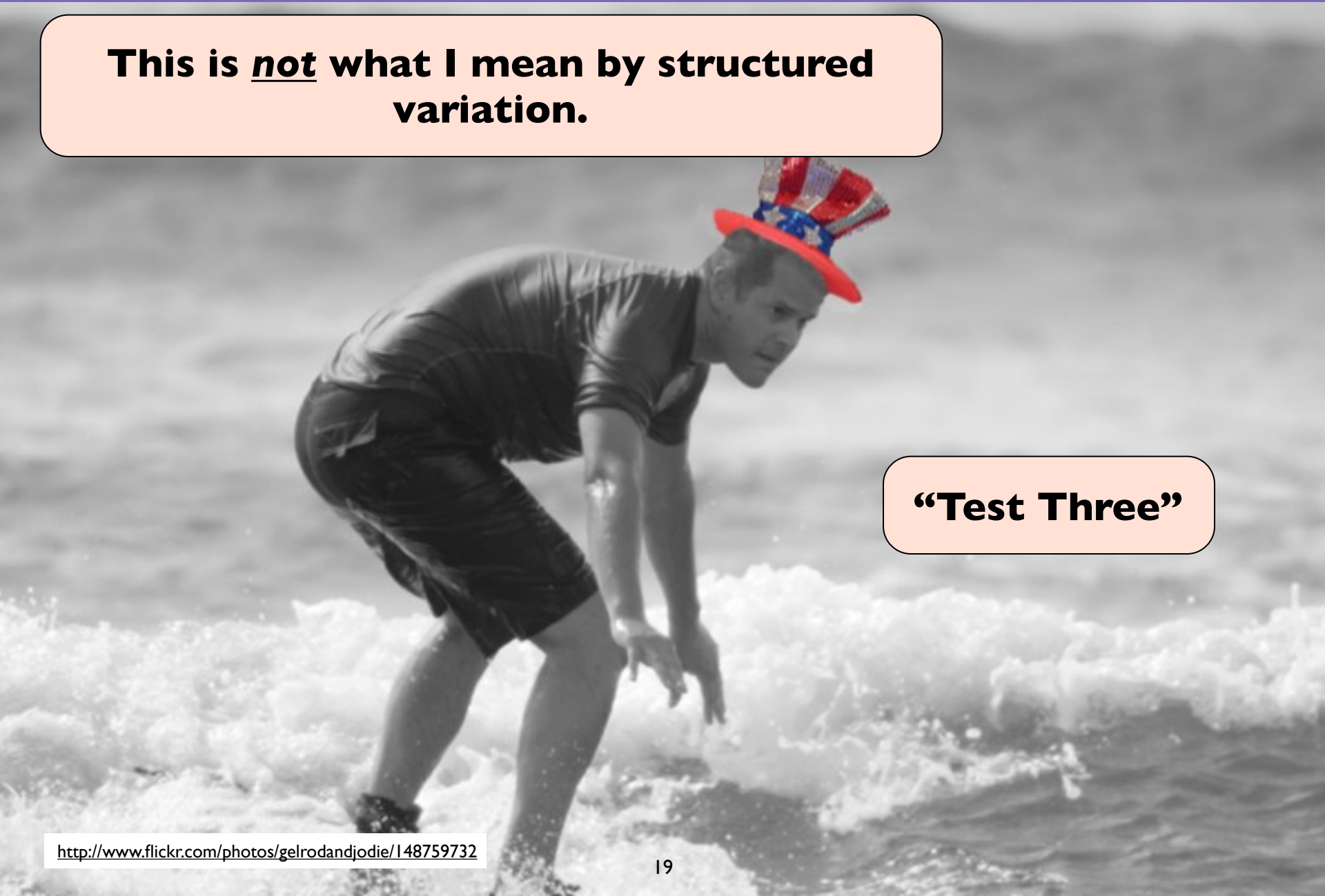


“Test Two”

http://www.flickr.com/photos/gelrodandjodie/148759732

Structured Variation?

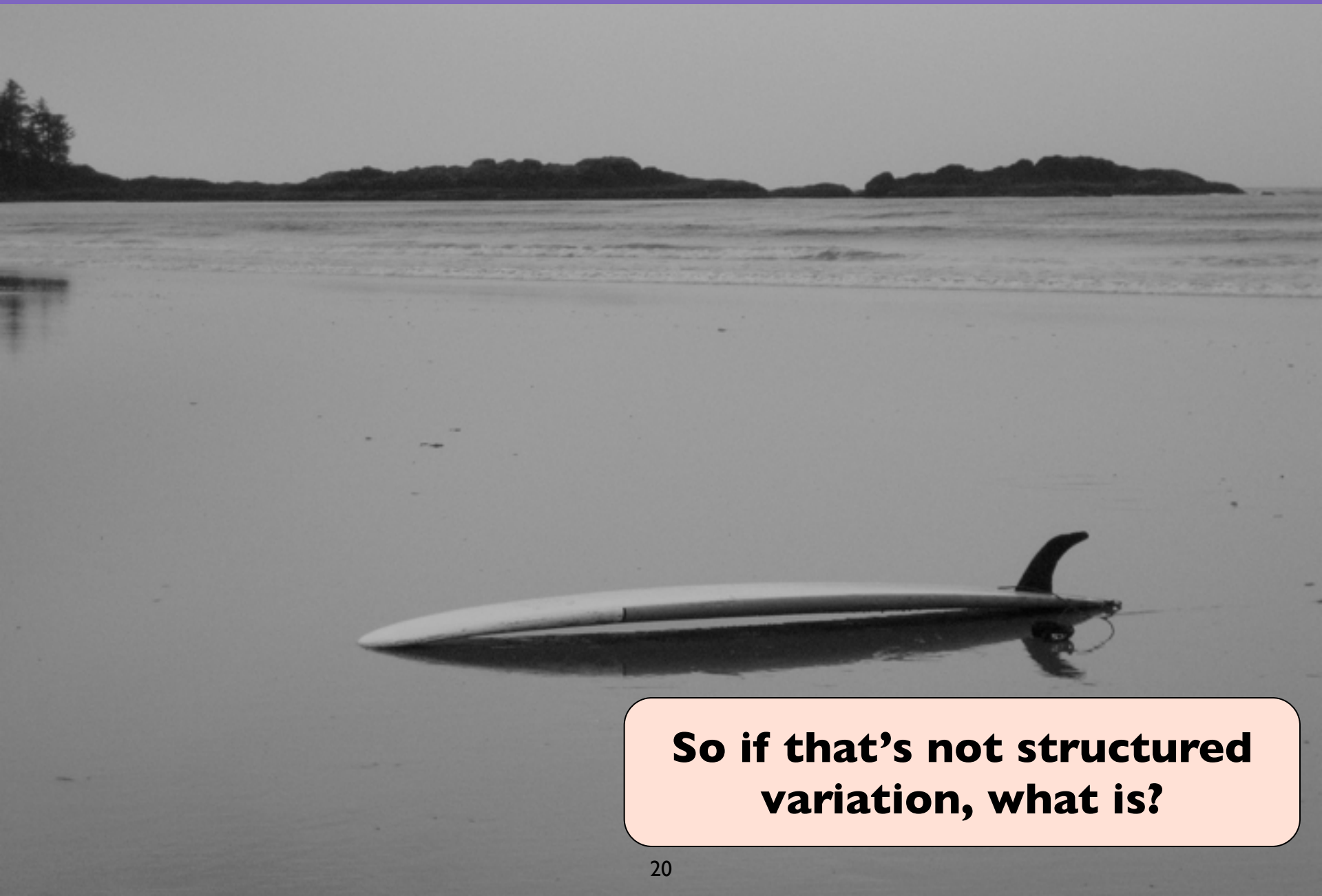
This is *not* what I mean by structured variation.



“Test Three”

http://www.flickr.com/photos/gelrodandjodie/148759732

Structured Variation?



So if that's not structured variation, what is?

Structured Variation

http://www.flickr.com/photos/tk_five_0/2281768728

Structured Variation

...each test will be different

http://www.flickr.com/photos/criben/2635618391

Structured Variation

http://www.flickr.com/photos/horrgakx/3186982135

Structured Variation

...testers will do their testing in different ways,

http://www.flickr.com/photos/pixelrobber/2626213081

Structured Variation

... and explore from more angles.

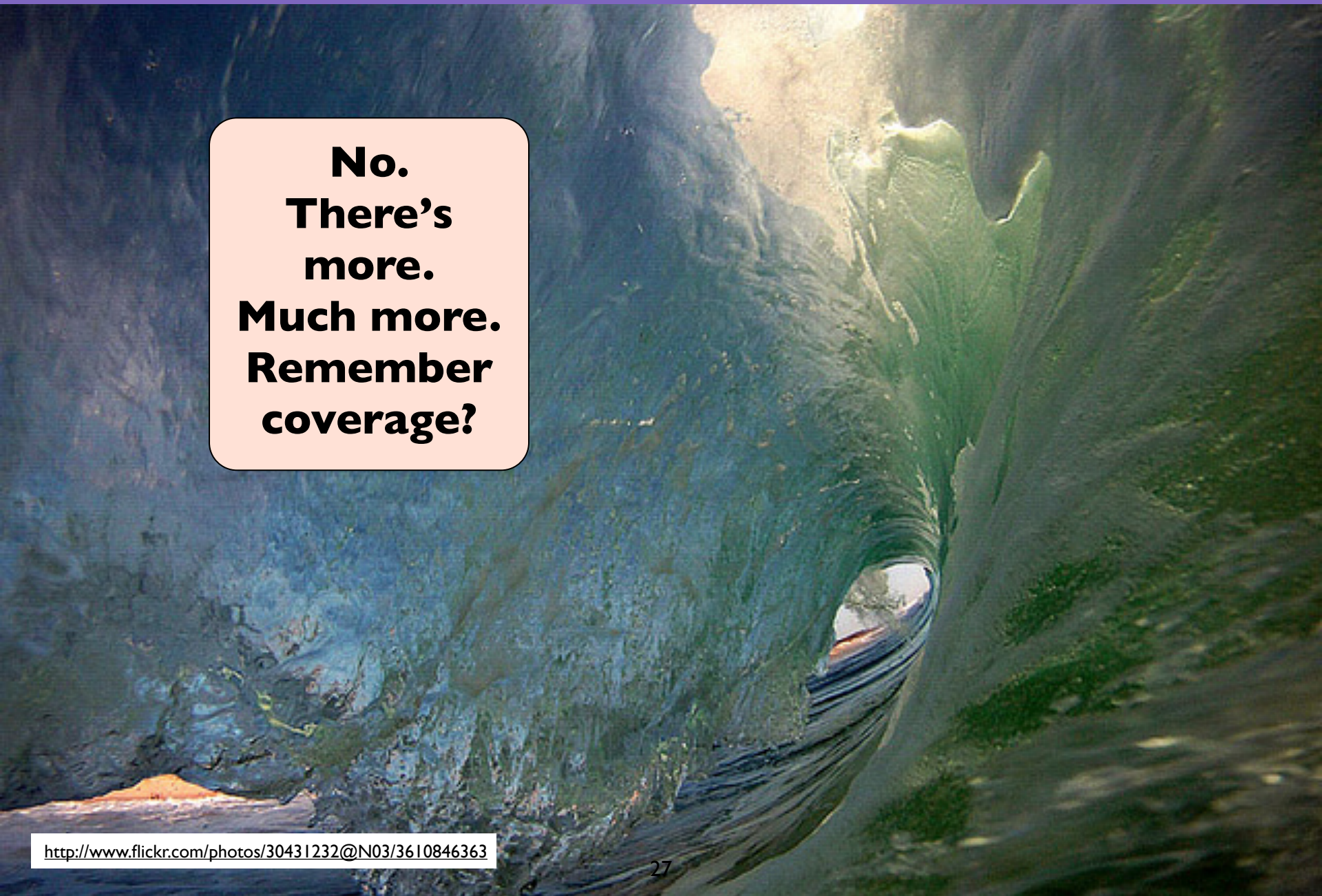
http://www.flickr.com/photos/tk_five_0/2286693552

Structured Variation

"OK. Maximum variation is better than redundantly repeated redundant repetition. Got it. That it? Am I good to go now?"

http://www.flickr.com/photos/johncarleton/4890241530

Coverage



http://www.flickr.com/photos/30431232@N03/3610846363

Coverage

Accidentally leaving gaps in coverage can lead to painful results.

http://www.flickr.com/photos/dwfromthepeg/240921271/

Coverage

http://www.flickr.com/photos/localsurfer/38245658/

Coverage

http://www.flickr.com/photos/mikebaird/4177570453/

Coverage

http://www.flickr.com/photos/dannohung/332507807

FOCUS OF SOLUTION & RATIONALE

Single-Mode Faults

Most software defects in production today can be triggered by just one test input.

http://www.flickr.com/photos/vaguelyartistic/292083492

Dual-Mode Faults

Software bugs caused by the interaction of two test inputs are very common too.

http://www.flickr.com/photos/kanaka/1798327442

Triple-Mode Faults

Defects that require three specific test inputs to trigger them are rare.

http://www.flickr.com/photos/kanaka/1798327442

Quadruple-Mode Faults

Defects that require four more test inputs to trigger them are extremely rare.

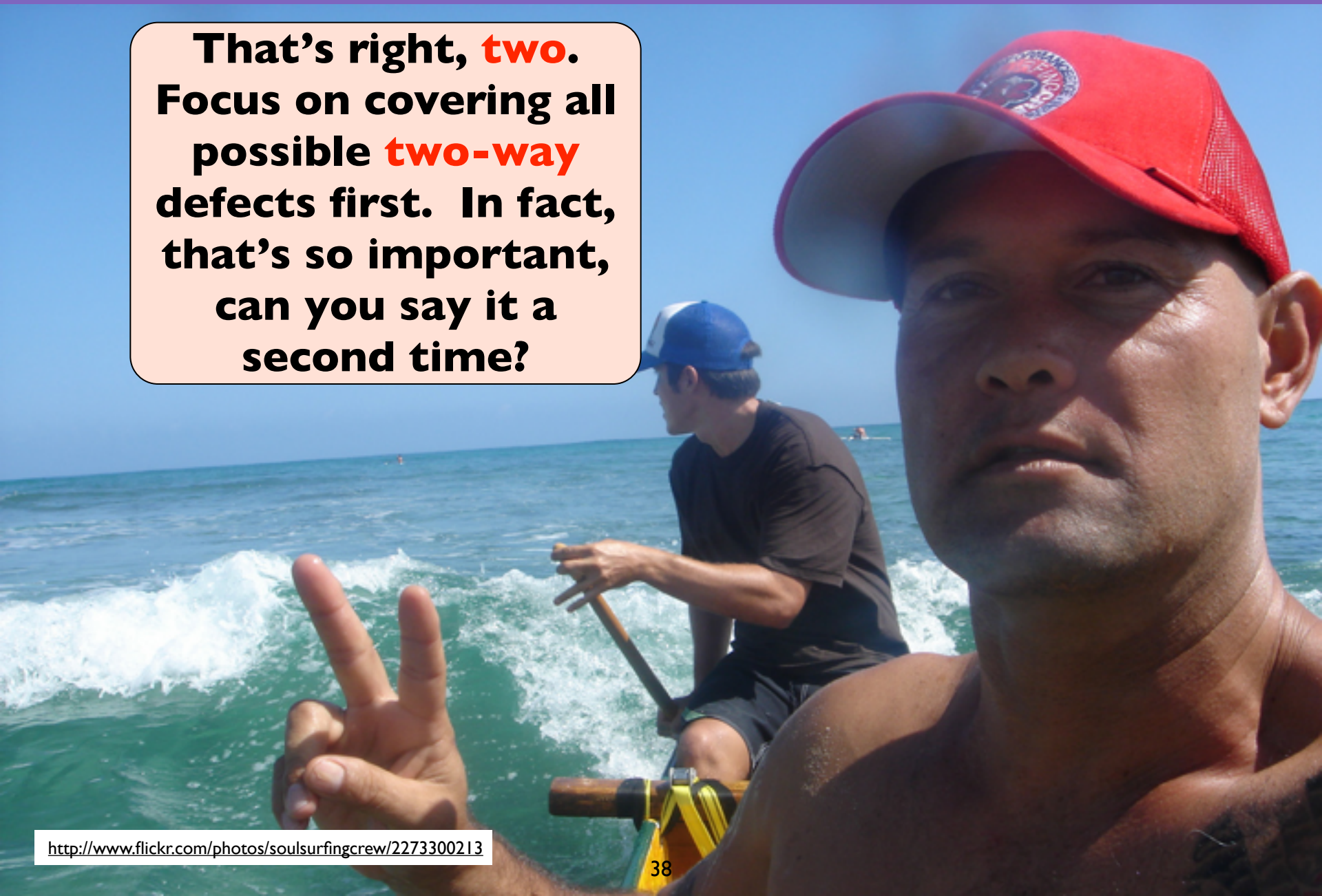
http://www.flickr.com/photos/novembering/3816524907

Implication on Testing Strategy

Aha! So, to find bugs efficiently, we should focus first on testing <u>all possible combinations</u> every set of *TWO test inputs!*

http://www.flickr.com/photos/johncarleton/4890236762

Implication on Testing Strategy



http://www.flickr.com/photos/soulsurfingcrew/2273300213

Implication on Testing Strategy

http://www.flickr.com/photos/johncarleton/4890236762

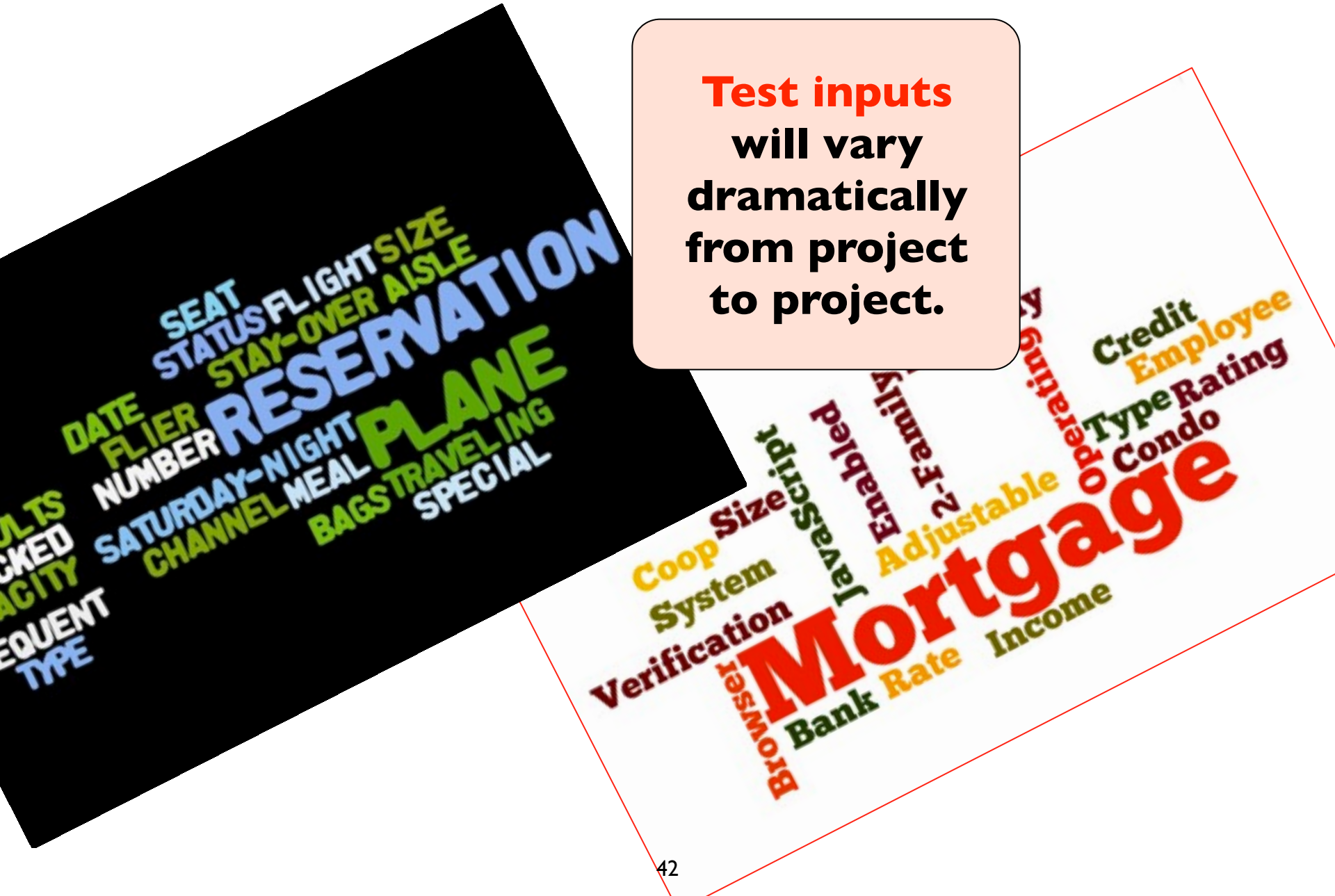
MECHANICS

Combinations of Test Inputs

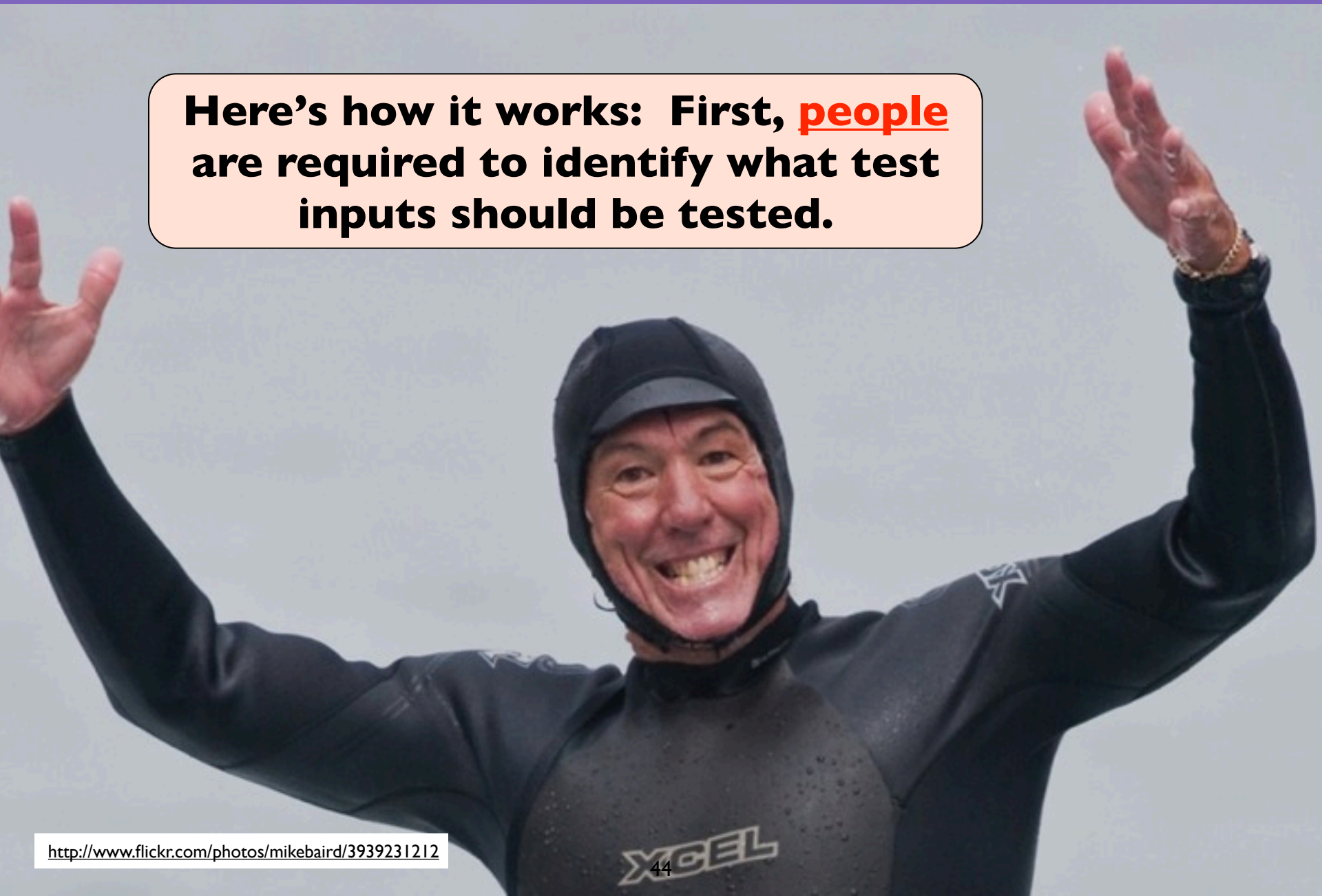
http://www.flickr.com/photos/12tribu/2750040244

Combinations of Test Inputs

Test inputs will vary dramatically from project to project.

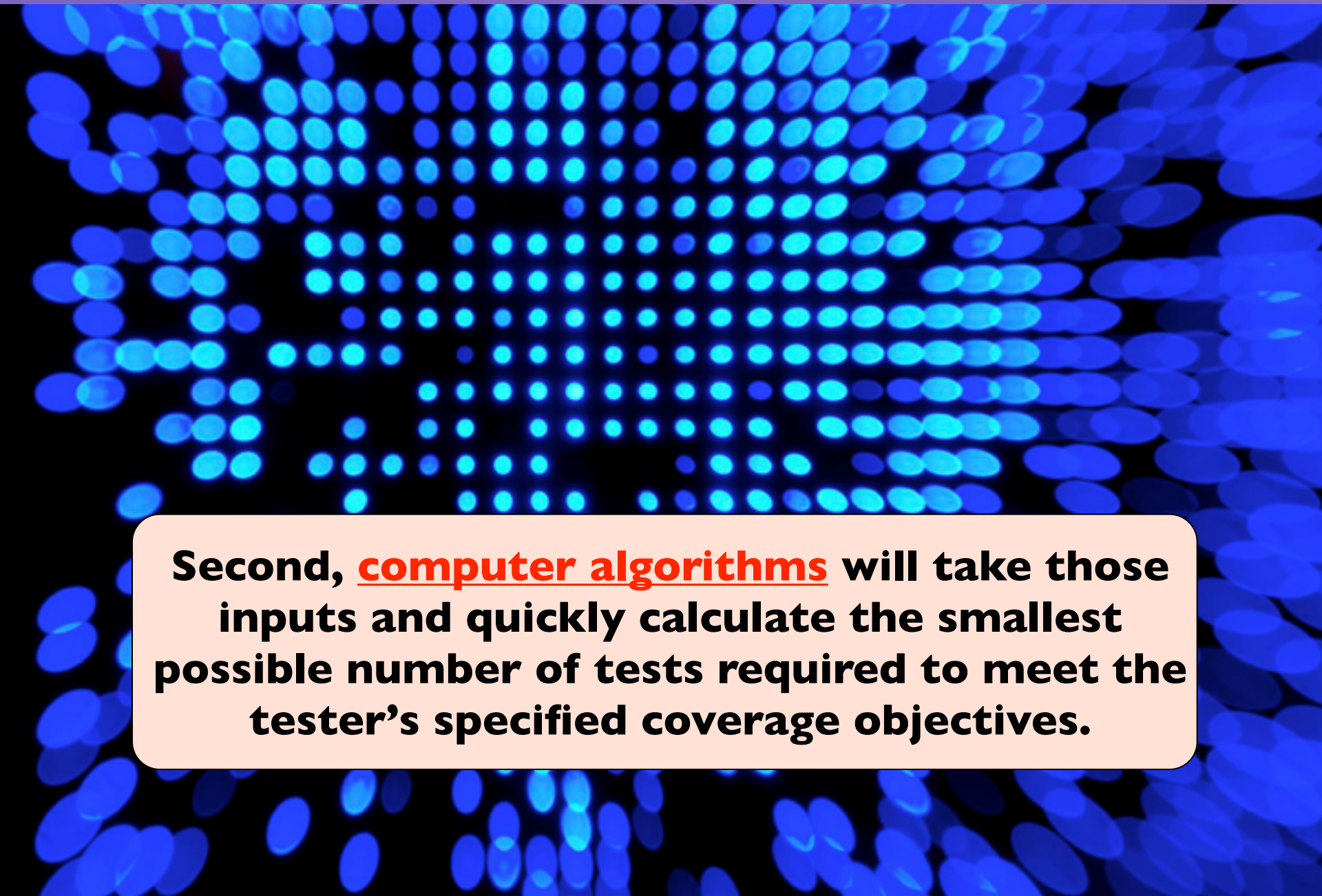


http://www.flickr.com/photos/johncarleton/4377591743



Here's how it works: First, people are required to identify what test inputs should be tested.

http://www.flickr.com/photos/mikebaird/3939231212



Second, computer algorithms will take those inputs and quickly calculate the smallest possible number of tests required to meet the tester's specified coverage objectives.

... and again, the algorithms will create *different solutions* depending upon your testing objectives.

http://www.flickr.com/photos/thomashawk/2402194148

http://www.flickr.com/photos/thomashawk/2402190314

Combinatorial Software Test Design

http://www.flickr.com/photos/thomashawk/2402194148

The woman wanting fewer tests should create 2-way tests

(AKA "pairwise" or "AllPairs" tests)

Combinatorial Software Test Design

This guy, wanting more coverage, should create more thorough combinatorial tests. (e.g., 3-way, 4-way, or multi-strength tests).




http://www.flickr.com/photos/thomashawk/2402190314

WHY NOT DESIGN TESTS BY HAND?

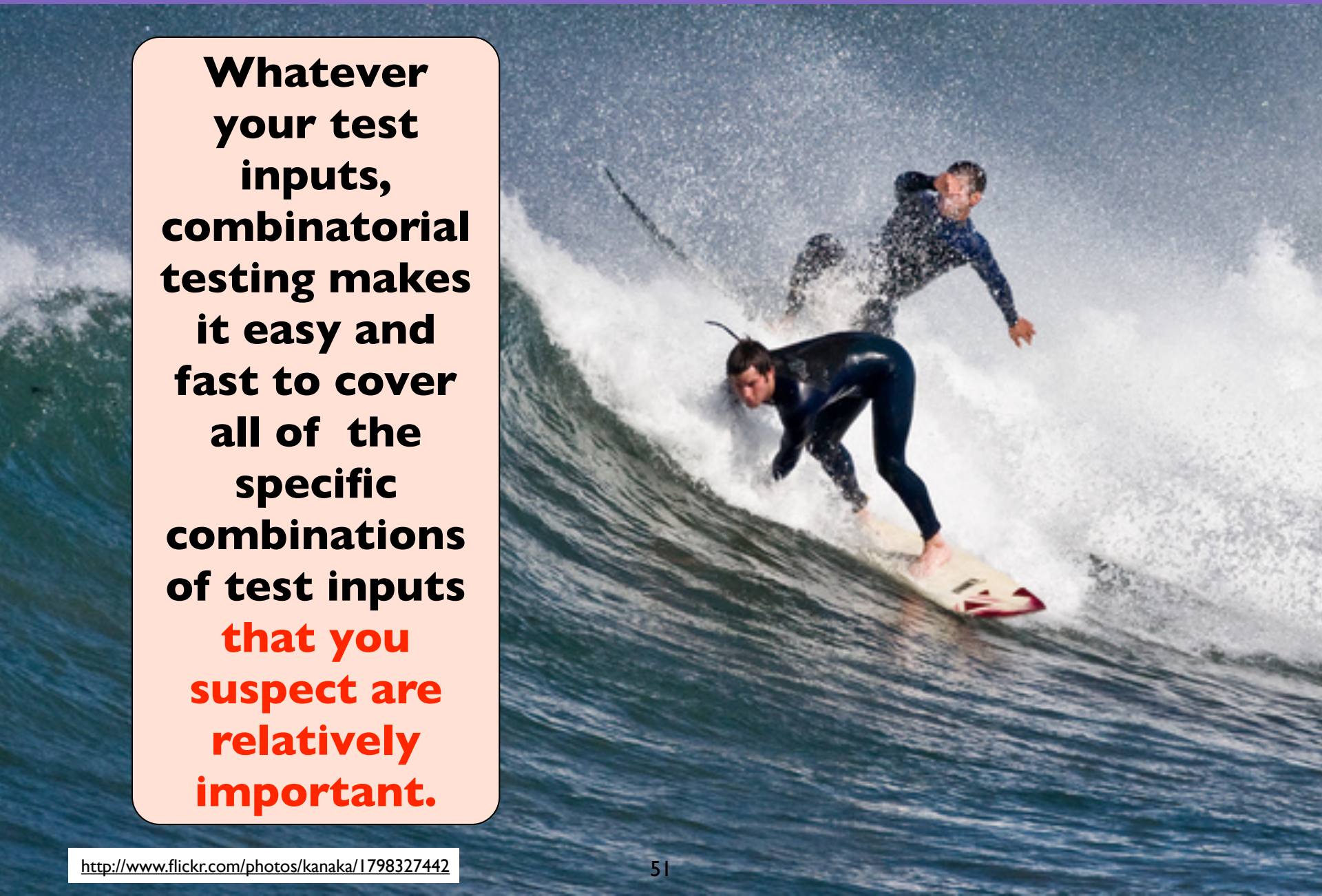
Advantages of Combinatorial Software Test Design

Test Design: Man vs. Machine
(Once test inputs are known)

	Speed:	Coverage:	Efficiency:
Human 	Weeks 	Gaps 	Way More Tests 
Computer 	Seconds 	100%* 	Way Fewer Tests 

*100% of what? 100% of the desired test input combinations (e.g., all pairs of values for 2-way tests solutions, all triplets of possible values for 3-way solutions, etc.).

Coverage Advantages



Whatever your test inputs, combinatorial testing makes it easy and fast to cover all of the specific combinations of test inputs that you suspect are relatively important.

http://www.flickr.com/photos/kanaka/1798327442

Advantages of Combinatorial Software Test Design

http://www.flickr.com/photos/johncarleton/4890239248

Advantages of Combinatorial Software Test Design

Perhaps, but maybe you...

Forget	Run out of time
Use more tests than needed	Accidentally repeat yourself

Coverage Advantages

***Plus*, using combinatorial testing, you'll *also get* additional coverage from combinations that wouldn't have made your list, including...**

http://www.flickr.com/photos/tk_five_0/2706278432

Coverage Advantages

http://www.flickr.com/photos/subspace-eddy/2766810770/

Coverage Advantages

... unexpected combinations,

http://www.flickr.com/photos/funky64/2875860978

Coverage Advantages

...notable, combinations that capture your attention,

http://www.flickr.com/photos/brb_photography/3221049399

Coverage Advantages

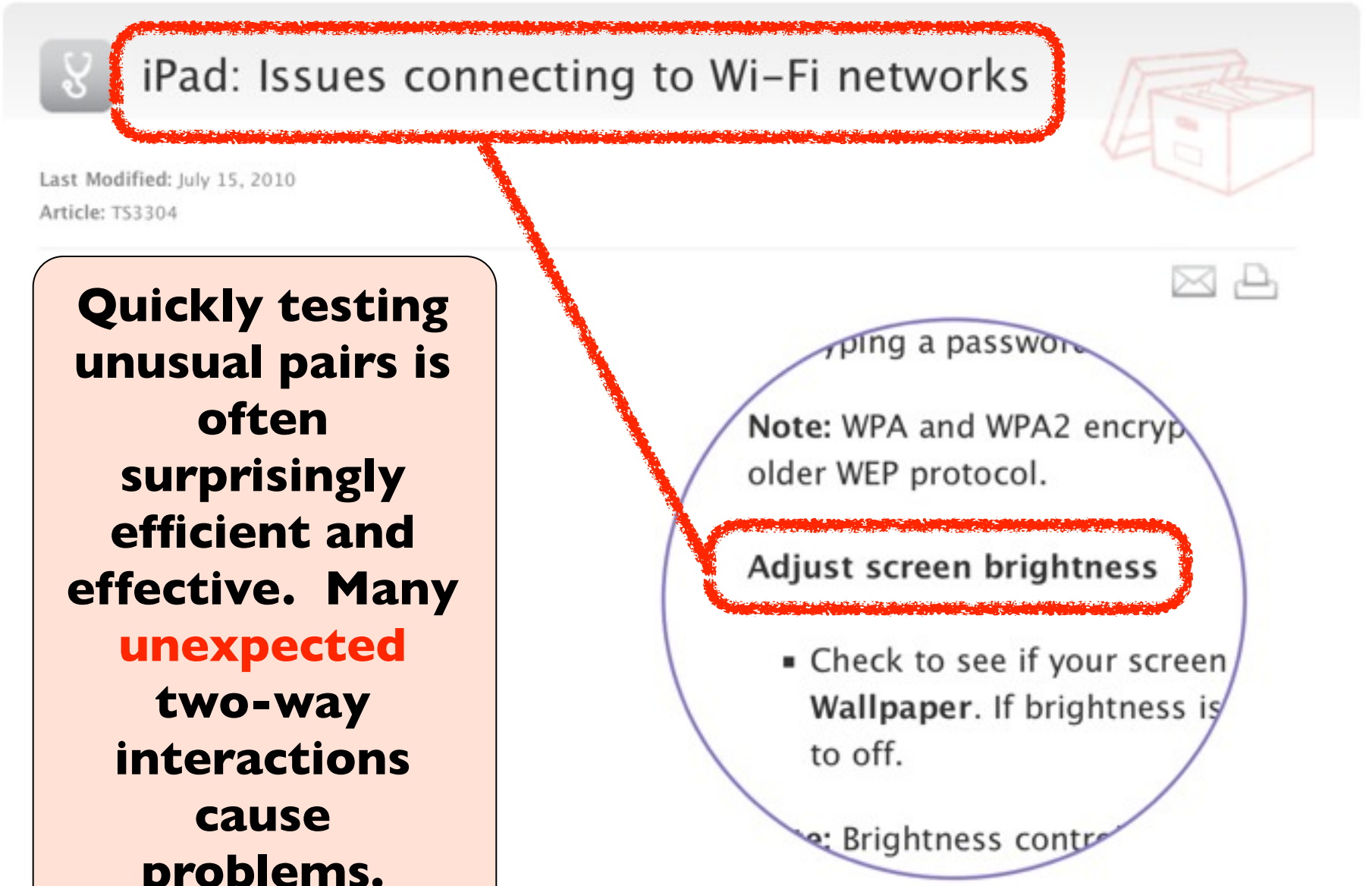
http://www.flickr.com/photos/grassvalleylarry/5536758

Coverage Advantages

... and combinations that will take you into different parts of the application (and different states) than you would have otherwise thought to explore.

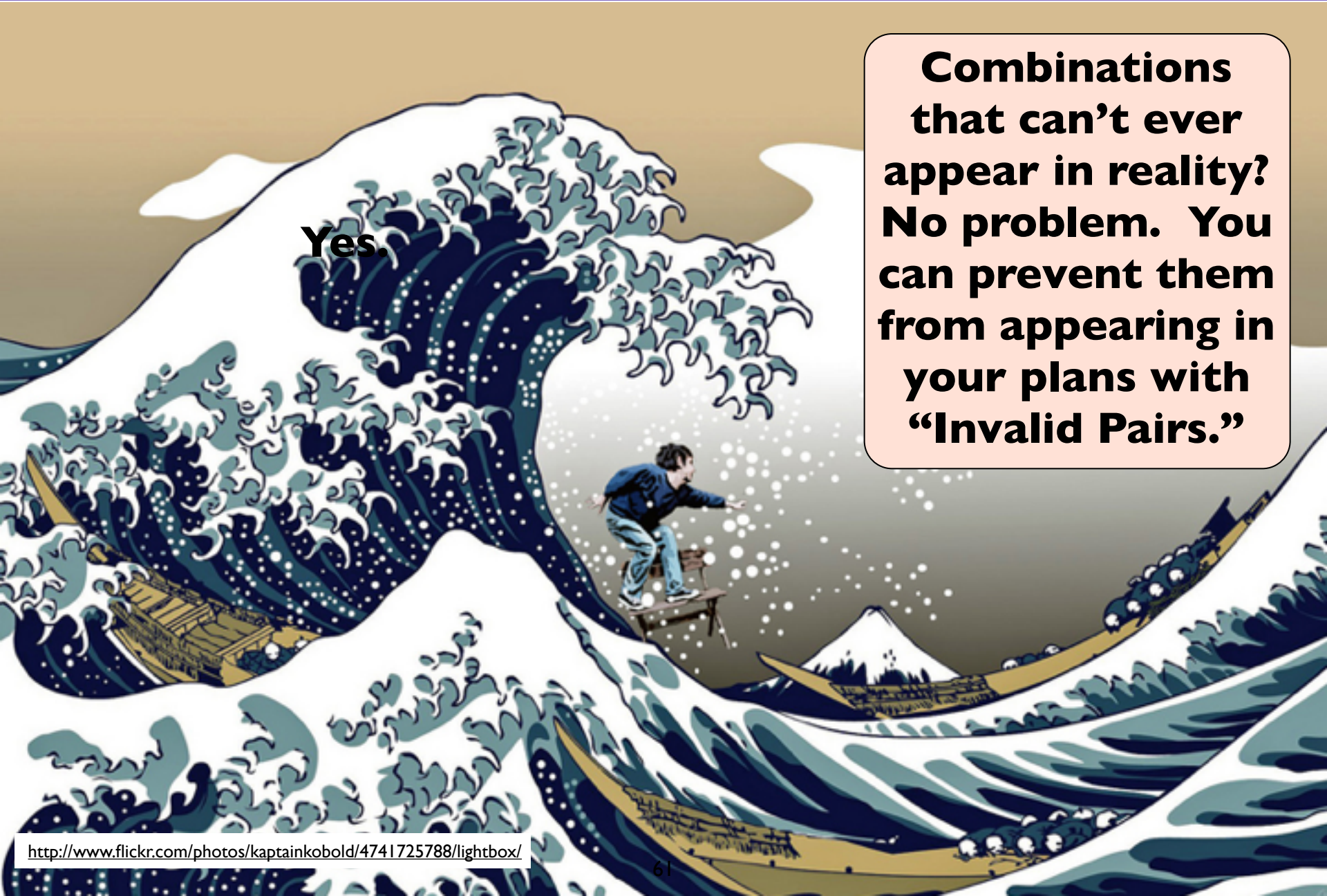
http://www.flickr.com/photos/28041049@N00/227352411

Coverage Advantages



Quickly testing unusual pairs is often surprisingly efficient and effective. Many unexpected two-way interactions cause problems.

Constraint Handling



http://www.flickr.com/photos/kaptainkobold/4741725788/lightbox/

READY YET?

Ready?

http://www.flickr.com/photos/johncarleton/4890241530

Hazards

Not so fast. Hazards exist. To succeed, you must use common sense.

http://www.flickr.com/photos/christianhaugen/3604106558

Limitations

And let's not kid ourselves. Not just anyone can do this stuff.

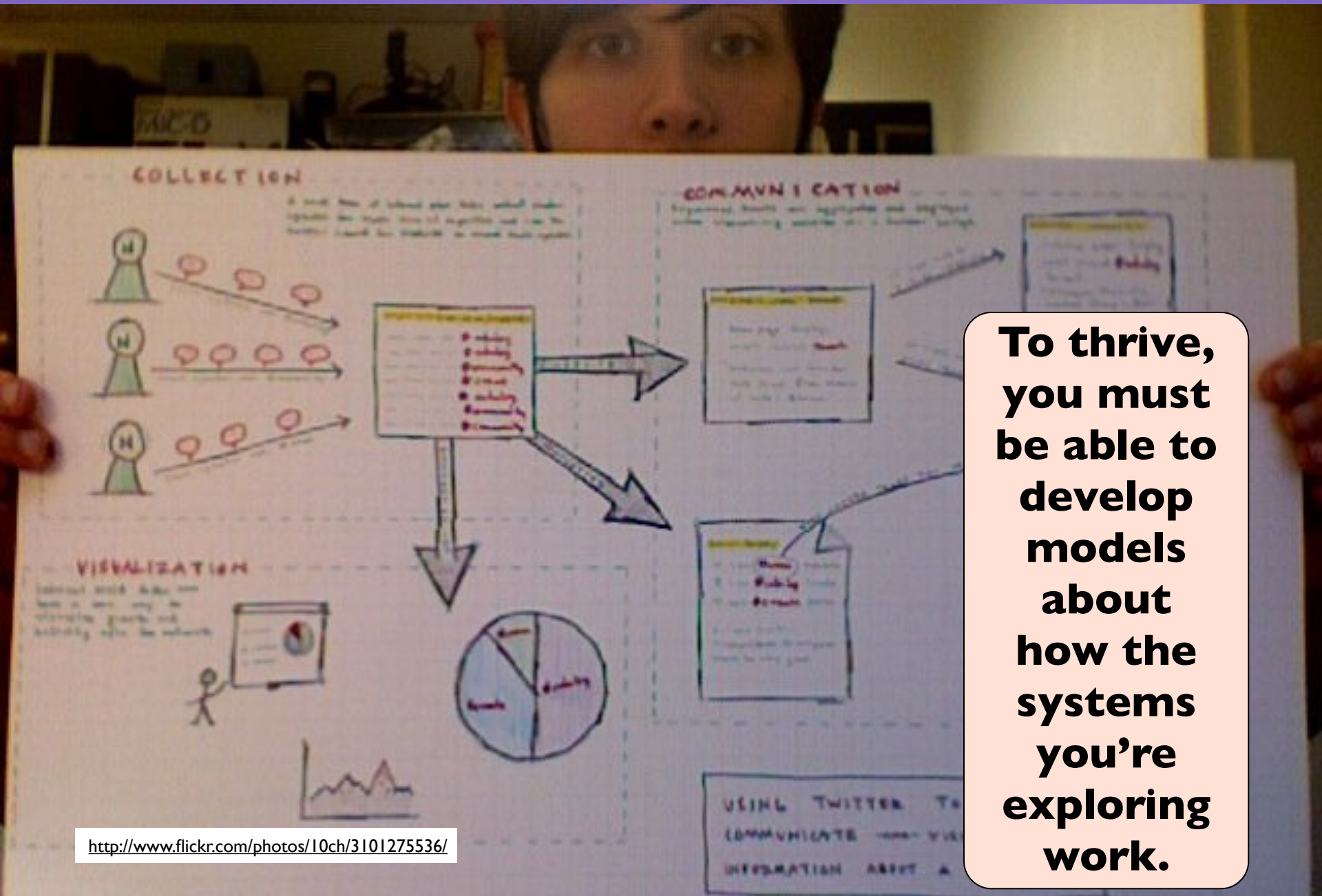
http://www.flickr.com/photos/mikebaird/573368639

Prerequisites

It also requires good critical thinking skills.

http://www.flickr.com/photos/johncarleton/9511759/

Prerequisites



To thrive, you must be able to develop models about how the systems you're exploring work.

http://www.flickr.com/photos/10ch/3101275536/

Practice Makes Perfect

Questions

http://www.flickr.com/photos/nathaninsandiego/3647136579

Ready?

http://www.flickr.com/photos/mikebaird/2996548709

http://www.flickr.com/photos/dannohung/332507807

Thank you

Special thanks to the folks at: flickr and its talented contributing photographers, including: Mike Baird

... and the inimitable John Carleton

(who, let the record show, had no input into the cartoonish statements we added next to his striking expressions).

Why 2-way Testing is Effective

Multiple thorough studies show approximately 85% of defects in production could have been detected by simply testing all possible pairs of values.

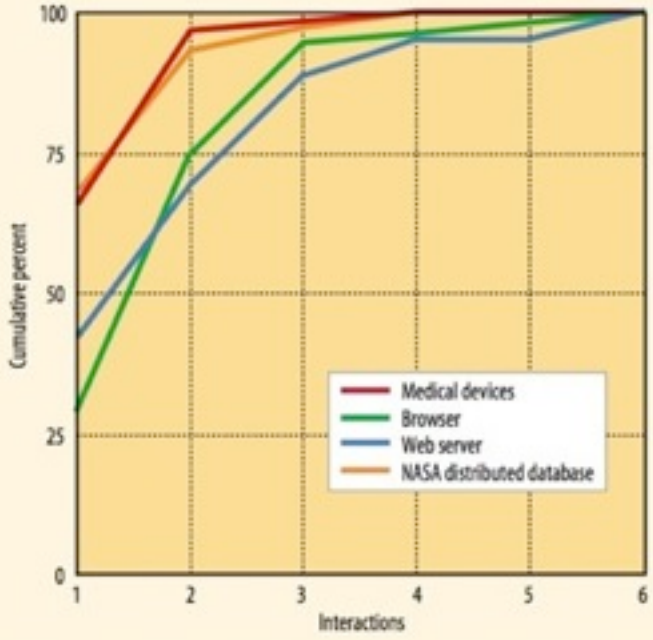


Figure 2. Cumulative error detection rate for fault-triggering conditions. Many faults were caused by a single parameter value, a smaller proportion resulted from an interaction between two parameter values, and progressively fewer were triggered by three-, four-, five, and six -way interactions.

Source: "Combinatorial Testing" IEEE Computer, Aug, 2009. Rick Kuhn, Raghu Kacker, Jeff Lei, Justin Hunter

Additional Information

Excellent introductory articles and instructional videos: www.CombinatorialTesting.com

Results of a ten-project study of effectiveness:

https://www.hexawise.com/Combinatorial-Softwar-Testing-Case-Studies-IEEE-Computer-Kuhn-Kacker-Lei-Hunter.pdf

Free trials of our firm's combinatorial test design tool: www.hexawise.com/users/new (which stay free indefinitely for companies using 1-4 licenses).

Also, please feel free to contact me if you have any questions. I'd be happy to quickly review a test plan or two, answer your questions, give quick pointers to help you run a pilot, etc. Seriously.... I enjoy helping people get started with this test design approach. Please don't hesitate to reach out. There's no charge and no catch.

justin . x . hunter @ hexawise . com

Four-Step Process to Design Efficient and Effective Tests

1. Plan

Scope

Be clear about single or multiple:

- Features / Functions / Capabilities
- User types
- Business Units
- H/W or S/W Configurations

Level of Detail

Acceptable options:

- High level "search for something"
- Medium level "search for a book"
- Detailed "search for 'Catcher in the Rye' by its title"

Passive Field Treatment

Distinguish between important fields (particularly those that will trigger business rules) and unimportant fields in the application.

Quickly document what your approach will be towards passive fields. You might consider: ignore them (e.g., don't select any Values in your plan) or a 3 Value approach such as "valid" "Invalid (then fix)" and "Blank (then fix)"

2. Create

Configurations

First add hardware configurations

Next add software configurations

Users

Next, add multiple types of users (e.g., administrator, customer, special customer)

Consider permission / authority levels of admin users as well as business rules that different users might trigger

Actions

Start with Big Common Actions made by users

After completing Big Common Actions, circle back and add Small Actions and Exceptions

Remember some actions may be system-generated

3. Refine

Business Rules

Select Values to trigger bus. rules

Identify equivalence classes

Test for boundary values

Mark constraints / invalid pairs

Gap Filling

Identify gaps by analyzing different states, orders of operations, decision-tree outcomes sought vs. delivered by tests, "gap hunting" conversations w/ SME's, etc.

Fill gaps by either (i) adding Parameters and/or Values or (ii) creating "one-off" tests.

Iteration

Refine longest lists of Values; reduce their numbers by using equivalence classes, etc.

Create Tests with and w/out borderline Values; consider cost/benefit tradeoffs of additional test design refinements

Consider stopping testing after reaching ~80% coverage

Consider 2-way, 3-way, and Mixed-Strength options

4. Execute

Auto-Scripting

Add auto-scripting instructions once; apply those instructions to all of the tests in your plan instantly

Don't include complex Expected Results in auto-scripts

Expected Results

Export the tests into Excel when you're done iterating the plan

Add complex Expected Results in Excel post-export

Continuous Improvement

If possible measure defects found per tester hour "with and without Hexawise" and share the results

Add inputs based on undetected defects

Share good, proven, plan templates with others

Practice Tips: Warning Signs

"You might be headed for trouble if..."

1. Plan

Scope

... You cannot clearly describe both the scope of your test plan and what will be left out of scope.

Level of Detail

... Parameters with the most Values have more Values than they require. 365 values for "days of the year" is bad. Instead, use equivalence class Values like "weekend" & "weekday." When in doubt, choose more Parameters and fewer Values.

Passive Field Treatment

... You cannot clearly describe your strategy to deal with unimportant details. If Values will impact business rules, focus on them. If Values don't impact business rules, consider ignoring them.

2. Create

Configurations

... You have ignored hardware and software configurations without first confirming this approach with stakeholders.

Users

... You have not included all the different types of users necessary to trigger different business rules. What user types might create different outcomes? Authority level? Age? Location? Income? Customer status?

Actions

... You start entering Small Actions (e.g., "search for a hardback science book by author name") before you enter Big Actions (e.g., "Put Something in Cart. Buy it.") First go from beginning to end at a high level. After you've done that, feel free to add more details.

3. Refine

Business Rules

... You forget to identify invalid pairs. - or - ... You rely only on Functional Requirements and Tech Specs w/out thinking hard yourself and asking questions to SME's about business rules and outcomes that are not yet triggered.

Gap Filling

... You assume that the test conditions coming out of Hexawise will be 100% of the tests you should run. There might well be additional "one-off" things that you should test and/or a few negative tests to design by hand.

Iteration

... You forget to look at the Coverage Analysis charts. If you achieve 80% coverage in the first quarter of the tests, you should measure the cost/benefit implications of executing the last 3/4 of the tests.

4. Execute

Auto-Scripting

... You add detailed Expected Results in the tests. - or - ... You forget that this feature exists and find yourself typing out test-by-test instructions one-by-one.

Expected Results

... You invest a lot of time in calculating and documenting Expected Results before you have determined your "final version" Parameters and Values. Last minute additions to inputs will jumble up test conditions for most test cases.

Continuous Improvement

... You don't ask (when defects that the tests missed are found post-testing) "What input could have been added to the test plan to detected this?" "Should I add that input to the Hexawise test plan now to improve it in advance of the next time it is used?"