A ~~hitchhiker's~~ software tester's guide to randomised testing - Part 2

22 May 2018 · agile, critical thinking, exploratory testing, random

How would test a water sac? (Wow there, calm that tester brain… I know what you are thinking, Whats it used for? Who / what uses it? how long does it need to last? Does the temperature of the water matter? Is it single use? etc. But let’s assume a generic hiking or camping water sac for now) I’m guessing one of your suggestions includes filling it with water, shaking it a bit and checking for leaks.

Seems kind of obvious right? but when it comes to software, we often do away with old-fashioned techniques such as filling something up and looking at it. Where’s the machine learning test algorithm? Call this a BDD scenario? Can Selenium check for H_20? I have to run this past the B.A…

(https://thumbs.dreamstime.com/b/water-leaking-bucket-30848081.jpg)

This is your software.

We can treat randomly generated test data and inputs in much the same way as water. Data files or other inputs like user interactions are the ever-moving parts of our applications. Think about it, the code is entirely static - it’s the state or data that is changing. This contradicts the commonly held notion that the code is like a set of complicated cogs churning away until they click into place on the answer.

If your app processes data files, JSON / XML feeds or some other kind of prescribed format of inputs - create one yourself, don’t just rely on the examples a Business Analyst or Product Owner has given you. Some hand rolled examples will probably be essential for smoking out bugs. But a large set of randomly generated data will probably also flush out some more. Much as you could easily detect the wet patches from a leak, you can often check easily for where your randomised data causes problems in your application.

With one client I created a large spreadsheet that included randomised data for a subset of the data fields. The generated data all matched the documented specification. Despite essentially being garbage, the data had valid relationships, text values and numbers. I imported this data into the system and looked at the user interface. What did I see? A sea of red! It turned out that the UI validation did not match the spec - in several places. Ad hoc testing had previously only found the more obvious validation mistakes.

The real smarts here probably isn’t the generated data file, though that made it easier to do this sort of analysis in bulk on a complicated app full of different data types and calculations. The clever bit here is: using the app to help check itself. The data fields that allow negative fields as per the spec, were highlighted in red in the UI when negative values were imported from our file. This made finding the inconsistencies in the validation easy. The random data helped us to try out things we might never have managed to think up for ourselves.

The same goes for a complicated series of user interactions. Let’s say the user works through a series screens as they buy or choose a product. Each screen has options that can take you to an array of different options. (The same principle applies to a complicated series of API interactions.). Using a set of BDD scenarios with data tables isn’t going to check many of the routes users are likely to follow.

Why not make random choices from valid options (you could also include invalid ones) and see what happens. You don’t have to know in advance what needs to happen exactly, just think of some heuristics. For example, If you get a 404 that’s probably a bug - take a closer look. You don’t even need to check the UI for that error you could check in the applications log. (Especially useful if it tries to hide the problem from the user with a helpful message / incorrect response code.)

A simple script to pick random values from a pull-down menu to reach the next screen and repeat can run in an infinite loop between each code check-in. Over and Over. Ever vigilant for

Javascript errors, error response codes, slow pages etc. Let the machine do the grunt work. Save the smart and detailed investigation for people.