Property based Testing aka PBT

by Jörg Gellien - 2022

Motivation

Testing is fun!

- a lot of experience in writing example based tests
- most of the time easy to reason about
- Missed out ideas and possibilities to verify properties or rules of system behaviour
- are there invariants within my function under test?
- E.g. if you sort a list the length of result stays the same ?
- in Java community Property Based Testing aka PBT not widely known but commonly used in functional languages like Haskell and Scala.

Rules of the Game

1. Specify rules or properties that the software under test should follow.
2. Verify this behaviour with generated test data like in the 100 or 1.000 instead of at most five examples like I used to do.

Following these simple rules opened up new point of view on my system under test. And it provided insight on expected behaviour I have not thought about before.

The Example Class

```
@Value
@With
public class
EventWrapper<T> {
  String eventId;
  String eventType;
  OffsetDateTime
eventTime;
  T data;
}
```

and the business event details.

```
@Value
public class
BusinessEvent {
  String
firstValue;
  String
secondValue;
}
```

- One wrapper class to hold common attributes for all events
- Generic connection to the business event details
- encode these event objects into and from the JSON representation

Why this Example

I stumbled over it during my working hours

- a common pattern in micro services architecture
- In recent years I developed services based on micro-services architecture.
- The communication patterns is based on messaging infrastructure using JSON messages.
- This pattern is reasonable well known in the audience and easy to follow along

The Demo

The Experiences so far

- There seems to be a problem with the provided type BusinessEvent.
- It found counter example on the first attempt - nice
- The counter example looks nicely reduced with interesting values I never planed to use in my production code
- What is this about Shrunken example and Original example?

The Shrinking Effect

finding the simplest representation of test data that reproduces an error

Like in a lot of other disciplines PBT shines when finding counter examples that reject the assertion of a property. PBT introduces the concept of shrinking trying to find the simplest example to reproduce the error. And this can ease up debugging and fixing the error a lot.

Demo Improved

What else did I found out

- With the changes in place there is just one thing left. Comparing the OffsetDateTime correctly.
  This problem never occurred to me in the first place and was not discovered with my initial examples.
- Now we have verified the behaviour or property with 1000 generated test data. And on every new run the next randomly generated test data will be used, until a counter example is found.

Patterns that are Designed for PBT

Based on ideas from Johannes Link paper "Patterns to Identify Properties"

Inverse Functions

Go back and forth

```
ForAll x:encode(decode(x))==x
```

Fuzzying

behave nicely all the time

Test Oracle

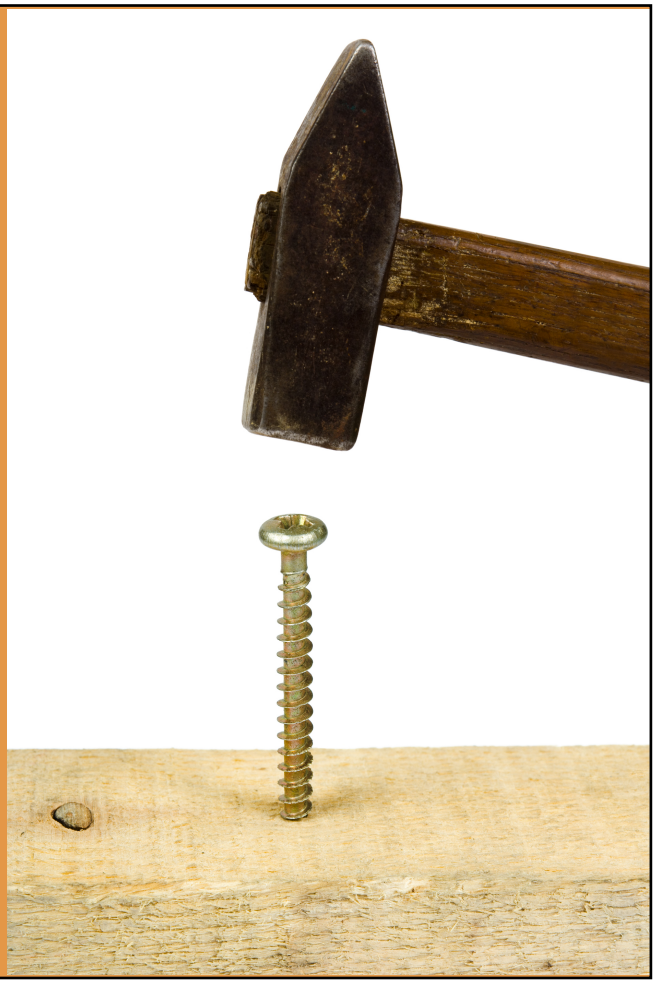
check old against new

```
ForAll x:old(x) == new(x)
```

There are a lot of functions that have a reverse. E.g. encode/decode JSON. So for all inputs Code should always be written that it will behave nicely. It should not spit into your face only because unexpected data examples. As examples you could check for:

- No exceptions occur, at least no unexpected ones - wrong data is identified on entering the service under test.
- No 5xx return codes in http requests, maybe you even require 2xx all the time
- All return values are valid There are a lot of functions and algorithms where you want to optimise. So you have to verify that This kind of verification is called test oracle.
  There are a few sources where the alternatives can come from:
- Simple and slow versus complicated but fast
- Self-made versus commercial
- Old (pre-refactoring) versus new (post-refactoring)

Lessons Learned



- Think outside the box. Get a deeper insight of the functioning of the system.
- On the other hand exploring new areas might lead to new ways of thinking and progressing in the realm of problem-solving. You can extend your repertoire of tools to use for solving different kind of challenges.

Things that came up during the experiments

- Am I restricting my input data correctly?
- will my system crash under some allowed but bizar input that never would come to my mind.

```
- Ever tried to use Chinese input characters?
```

- Definitely not for the faint of heart.
  You have to think hard to find properties that represent your system.
  And it can be challenging to describe data generators to prove specific properties.
- PBT will be a good solution if you find structures and algorithms where your gut feeling starts to moan and the existing tests do not provide sufficient security and understanding of the system.
- Myself I am astonished how many ideas and insights I generated only by providing some unexpected input data.

At the End

Exploring PBT led to

- a lot of fun
- a cool way to generate test data
- and a new tool for my belt

Questions or Remarks ?

References

- For all the examples I use jqwik provided by Johannes Link, who is well known in the java testing community. JQWIK project page
- The sample project with the code and my article with a lot more references you can find on github jghamburg/pbt-demo.
- The associated article is published on jghamburg.github.io
- Patterns to Identify Properties by Johannes Link 2018