Software Testing, Quality Assurance & Maintenance—Lecture 12

Patrick Lam
University of Waterloo

February 13, 2026

Part I

Property-Based Testing

Begging the Question. . .

Property-based testing is
. . . testing using
properties?

Go on…

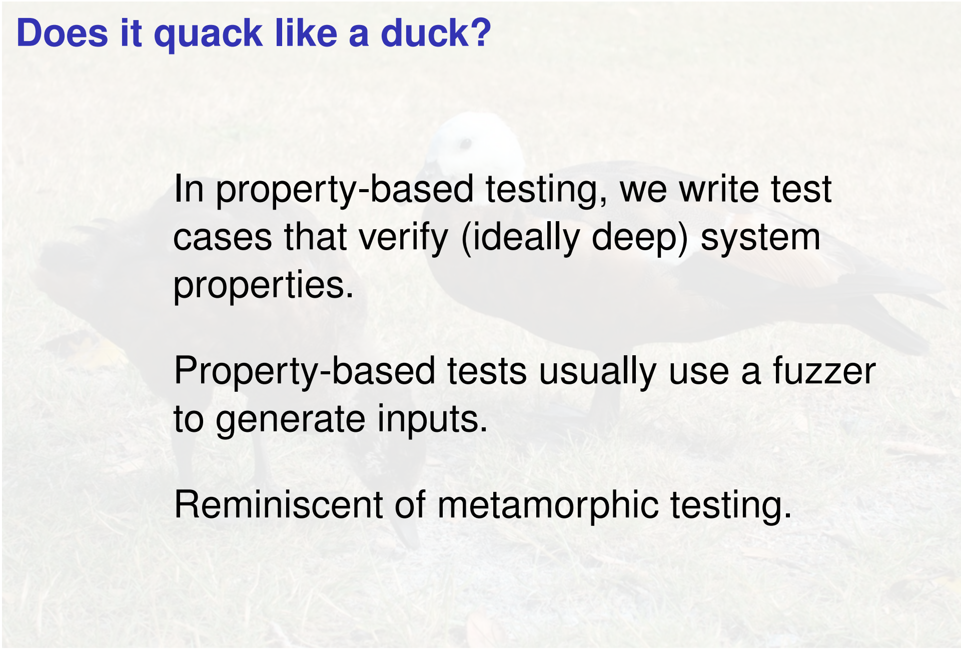
What kind of properties?

- specify the “shape” of “interesting” inputs;
- properties of the system’s abstractions;

Or: it triggers failures that could not have been revealed by direct fuzzing.

Or: properties are things that are true for any correct implementation.

Does it quack like a duck?



In property-based testing, we write test cases that verify (ideally deep) system properties.

Property-based tests usually use a fuzzer to generate inputs.

Reminiscent of metamorphic testing.

`map`

```python
def map(fn, xs):
  ys = []
  for x in xs:
    ys.append(fn(x))
  return ys
```

As we'd expect, `list(map(lambda x: x+1, [3, 4]))` yields `[4, 5]`.

An example property test

Python Hypothesis library:

```
@given(st.lists(st.integers()))
def map_identity_yields_self(xs):
    id = lambda x: x
    assert list(map(id, xs)) == xs
```

`map`ping the identity function $\lambda x.\ x$ on list ℓ yields the same list ℓ.

Note: not concrete

```python
@given(st.lists(st.integers()))
def map_identity_yields_self(xs):
    id = lambda x: x
    assert list(map(id, xs)) == xs
```

Hypothesis generates inputs (lists of integers) using fuzzing-like techniques.

It calls this function repeatedly with generated inputs.

Purpose: ensure that assertions hold.

What Hypothesis Does

Here's some skeleton F# code:

```
let propertyCheck property =
  // property has type: int -> int ->
     bool
  for _ in [1..100] do
    let x = randInt()
    let y = randInt()
    let result = property x y
    Assert.IsTrue(result)
```

Getting diagnostics

```
@given(st.lists(st.integers()))
@settings(verbosity=Verbosity.verbose)
def map_identity_yields_self(xs):
    id = lambda x: x
    assert list(map(id, xs)) == xs
```

Otherwise, no news is good news.

When you do get a failure report,
Hypothesis simplifies the report.

List Examples

In the full notes, you'll see a bunch of examples.

I won't put them on slides, but you can find them in `code/L12/list_tests.py`.

Patterns for Property-Based Testing

Everyone who sees a property-based testing tool like FsCheck or QuickCheck thinks that it is amazing but when it comes time to start creating your own properties, the universal complaint is: "What properties should I use? I can't think of any!"

Common Patterns

- Different paths, same destination
- There and back again
- Some things never change
- The more things change, the more they stay the same
- Solve a smaller problem first
- Hard to prove, easy to verify
- The test oracle

Different paths, same destination

Check that doing X then Y gives the same thing as doing Y and then X.

e.g. sort, add 1.

There and back again

Do X and then its inverse X^{-1}.

e.g. serialize/deserialize.

Some things never change

Is an invariant preserved?

e.g. collection size/contents

The more things change, the more they stay the same

Idempotence: doing an operation twice is same as doing it once.

e.g. deduplicating a collection

Solve a smaller problem first

Properties based on structural induction.

Hard to prove, easy to verify

e.g. maze pathfinding vs verification, factorization into primes, string tokenization vs concatenation, literally proof derivation vs verification.

Test oracle

Compare with results from an oracle.

Part II

Commentary

Property testing vs metamorphic testing

Pretty similar.

Properties in property-based testing will often qualify as metamorphic relations.

In property testing, use framework to generate tests; in metamorphic testing, use the relation to generate more tests.

Property testing vs fuzzing

Property testing uses fuzzing.
Fuzzing benefits from implicit oracles.

Difference: fuzzing focussed on creating random inputs & finding crashes (especially security problems).
In fuzzing, system properties are incidental.

Property-based testing relies on developer to describe interesting inputs.

Property testing and system design

Claim: property testing gives a better understanding of system design & invariants, as you are designing the system.

Like Test-Driven Development (TDD), but with deeper properties.
TDD: write minimal working code that passes the test cases.
Property testing: write minimal code (the EDFH code), then think of properties that break this code.