Property-based Testing in Practice

Yifeng He

Aug 24, 2024

Disclaimer

This empirical study is based on survey/interview, so our focus will be on discussing experiences and potential opportunities rather than proposing new methods or experiments.

Background: Property-based Testing (PBT)

Properties are executable specifications of programs. For example, a binary tree is a *valid* BST if and only if each internal node has data grater than any in its left subtree and less than any in its right subtree. Then, when inserting a new node to a valid BST, the BST must still be valid after the insertion.

```
@given(x=integers(), t=trees())
def test_insert_maintains_bst(x, t):
  if is_valid_bst(t):
    assert is_valid_bst(insert(t, x))
```

- Generators: `integers()` and `trees()` generates valid inputs of the corresponding type.
- Pre-condition: `if is_valid_bst(t):`
  - it simply discards trees until `trees()` happens to generate a valid BST
  - the *majority* of the generated trees will be discarded, wasting precious generation time
- Shrinking: If the property ever fails during testing, a shrank/reduced failing value is presented to the user.

Methodology

This study conducted 30 interviews with people in Jane Street (fin-tech company), where

- 26/30 are testers, who use PBT tools in their day-to-day work,
- 4/30 are maintainers, who play a role in building and maintaining PBT infrastructure.
- between 1 and 26 years of professional Software Engineering experience (median 7)
- between 1 and 20 years using PBT (median 6).

Benefits of PBT

- **Improves Code Correctness**: used for critical code to detect hard-to-find bugs.
- **Increases Confidence**: helps developers feel more assured about code quality.
- **Finds Unexpected Bugs**: often catches edge-case errors missed by other tests.
- **Enhances Understanding**: forces developers to clarify their thinking and catch misunderstandings.
- **Serves as Documentation**: properties act as persistent, executable documentation.
- **Encourages Adoption**: PBT's usefulness encourages wider use and support within teams.

Comparisons to Other Testing Approaches

v.s. example-based unit testing

PBT is more concise but less transparent than example-based tests.
Unit test is

- “often easier to understand”
- “explain what they’re doing to a better degree, easier to review”
- “just print out a bunch of crap, too much stuff.”

v.s. fuzzing

PBT is used during development with strict time limits; fuzz testing is used separately for integration tests.
Fuzzing is

- forgotten and left running for a year and a half on a spare server
- running “out of band”
- outside of the normal CI and at time scales on the order of hours or days
- fuzzing tools are given *much* longer to run than PBT tools.

Writing Specifications

Generating Test Data

Understanding Failure

Understanding Success

Observations

Research Opportunities