More Testing, Fewer Tests

Tyler A. Young

ElixirConf US
Orlando

@TylerAYoung

TylerAYoung.com

Let’s build a permissioning system!

Ability	Logged out	Viewer	Author	Editor
View published post	✔	✔	✔	✔
View draft post		✔	✔	✔
Create post			✔	✔
Edit own post			✔	✔
Edit others’ post				✔

Let’s see some code!

Test diffusion

Logic gets smeared across 4 × 5 tests

So much for tests as documentation!

Realistically, there could be 20–50 rows in this table 😱

Ability	Logged out	Viewer	Author	Editor
View published post	✔	✔	✔	✔
View draft post		✔	✔	✔
Create post			✔	✔
Edit own post			✔	✔
Edit others' post				✔

The dark side of testing

Tests are code, and code is a liability.

All else being equal, more tests are worse than fewer tests.

- Slower to write
- Slower to run
- More to update when changing behavior
- More to take in when reading tests as documentation
- More to check when deciding if an existing test case should have covered a situation you're debugging

Can we get the same (or better!) confidence from fewer tests?

Permissions are policies

- Arbitrary
    - The opposite of objective
    - Few invariants
- High likelihood of changing or extending in the future

What about property-based testing?

- PBT uses invariants and generated data to check behavior
  - Every member of a sorted list is greater than or equal to the one before it
  - Round-trip through an encode + decode process produces the same value you started with
- Not all code is heavy on invariants, though!

Invariants in the permissions system

- Permissions increase left-to-right
- Never have more permissions on drafts vs. published posts
- Owned posts never have less permissions than unowned

Ability	Logged out	Viewer	Author	Editor
View published post	✔	✔	✔	✔
View draft post		✔	✔	✔
Create post			✔	✔
Edit own post			✔	✔
Edit others' post				✔

Policies are not invariants

- Logged-in viewers can see draft posts
- There's a button on the home page labeled "Sign Up"
- The Free plan gets features *x* and *y*, but the Professional plan also gets feature *z*
- Trials are 14 days

Invariants: Relative datetimes

```
@spec relative_datetime(DateTime.t()) :: String.t()
```

- 30 seconds or less → “just now”
- < 60 minutes → “*x* mins ago”
- < 24 hours → “*x* hours ago”
- < 30 days → “*x* days ago”
- < 365 days → “*x* months ago”
- Else → “*x* years ago”

Invariants: Relative datetimes

```
@spec relative_datetime(DateTime.t(), DateTime.t()) ::
        String.t()
def relative_datetime(dt, now \\ DateTime.utc_now())
```

- 30 seconds or less → "just now"
- < 60 minutes → "*x* min(s)"
- < 24 hours → "*x* hour(s)"
- < 30 days → "*x* day(s)"
- < 365 days → "*x* month(s)"
- Else → "*x* years"

[Aside] Property test coverage in CI

By default, you don't get great exploration of the edges in your property tests.

You probably want more runs, potentially with a capped runtime instead of capped number of tests.

The dark horse: snapshot testing

What if we recorded the behavior of the function as-is
and compared future runs with this one?

Extremely useful when coming into a system with no tests

Libraries:

- assert_value (useful generically)
- heyya (for Phoenix LiveView)

You don't understand a tool until you know when not to use it.

Comparison of testing methods

	Barrier to entry	Speed to write	Maintenance & extension	Logical grouping	Communication
Traditional	Lowest	Slow	Linear	Low	Medium

Comparison of testing methods

	Barrier to entry	Speed to write	Maintenance & extension	Logical grouping	Communication
Traditional	Lowest	Slow	Linear	Low	Medium
Parameterized	Low	Slower	Sub-linear	High	High

Comparison of testing methods

	Barrier to entry	Speed to write	Maintenance & extension	Logical grouping	Communication
Traditional	Lowest	Slow	Linear	Low	Medium
Parameterized	Low	Slower	Sub-linear	High	High
Property	High	Slowest	Sub-linear	High	Depends

Comparison of testing methods

	Barrier to entry	Speed to write	Maintenance & extension	Logical grouping	Communication
Traditional	Lowest	Slow	Linear	Low	Medium
Parameterized	Low	Slower	Sub-linear	High	High
Property	High	Slowest	Sub-linear	High	Depends
Snapshot	Low	Fast	Linear	Low	Low

Comparison of testing methods

	Traditional	Parameterized	Property	Snapshot
Unconditional behavior	✔			
Low-impact behavior	✔			✔
Policies		✔	✔ (Double-check)	
Invariants			✔	
Low existing coverage				✔

Summary

- All else being equal, fewer tests are better
- Tools for writing fewer tests:
  - More assertions per test
  - For comprehensions within a test
  - Elixir 1.18 built-in parameterized tests
  - parameterized_test library
  - Property-based testing
  - Snapshot testing

Need Elixir development or consulting? Reach out!