Questions to ask yourself when writing code

Michal Charemza

Am I sacrificing the present for some “future”?

- Safety of current code is paramount
- **CR**: data flow clarity, not OO-modelling, fewer cases
- **QA**: code needs to run and have output
- **Tests**: need to exist and assert on output

- Very often it’s not a trade-off: clear and safe code **now** often means easy to change **later**

What would this test catch, and not catch?

- Each test defines a layer
  - Allows changes inside
  - Makes **zero** assertions on correctness of boundary
- Small layers often forbid working changes while allowing failing ones: exactly the opposite of what is desired
- For safety **today**, not just future

What would this test catch, and not catch?

- Special mention: Angular templates especially-unsafe
  - Directives silently don’t compile
  - (Most) directives don’t check arguments
  - Expressions often silently fail
- Often the DOM + (mock) HTTP is a reasonable boundary for browser-side tests

What would this test catch, and not catch?

```
describe("Model::getEndpoint", function() {
  it("is /api/model", function() {
    expect(Model.getEndpoint()).toBe("/api/model");
  });
});
```

What would this test catch, and not catch?

```
describe("user selects -> poll -> poll resolved -> selection resolved", function() {
  it("preserves the user selection", function() {
    expect(this.selection()).toBe('Alpaca')
    this.select('Llama');
    expect(this.selection()).toBe('Llama');
    this.poll();
    expect(this.selection()).toBe('Llama');
    this.pollResolve();
    expect(this.selection()).toBe('Llama');
    this.selectResolve();
    expect(this.selection()).toBe('Llama');
  });
});
```

Is the old behaviour a special case of new?

- … or can we make it be by refactoring?
- Special branches for old and new **adds** complexity
- “We only sometimes do it” ⇒ we do it!
- Refactoring first
  ⇒ new behaviour can often be a 1 line change
- E.g. *Sometimes* filter, refactor to *always* filter, but filter chosen dynamically

What if I remove branching?

- Fewer branches ⇒ easier to reason ⇒ fewer bugs
- Tests and QA hit more code ⇒ fewer bugs
- Optional arguments *are* branches
- Often runtime checks can be converted to static value

Can I move (impure) code out of branches?

- More consistency ⇒ fewer bugs
- Tests and QA hit more code ⇒ fewer bugs
- Or a step to collection pipelines + ternary operator (which *is* fine, in spite of Python syntax)
- Bonus: often easier to change later

Can I move (impure) code out of branches?

```
if something:
  doSomethingA()
else:
  doSomethingB()
```

```
value = \
  a if something else \
  b
doSomethingC(value)
```

Can I change/override fewer things?

- Setting something once & in one place

  ⇒ easier to reason about ⇒ fewer bugs

- Bonus: often easier to add features in future
- E.g. overriding base methods
- E.g. overriding CSS

Am I about to avoid something better but boring?

- 1 hour of dev time is small in the grand scheme

What if we didn’t use this library?

- Would it be faster/less complex/more flexible?
- Don’t assume random person on internet is better

What if my assumptions aren't satisfied?

- "Safety" is context-specific
- Often an exception *is* the best thing
- Make **consistent at source** rather than adding complexity to destination

Am I working with code that shouldn’t exist?

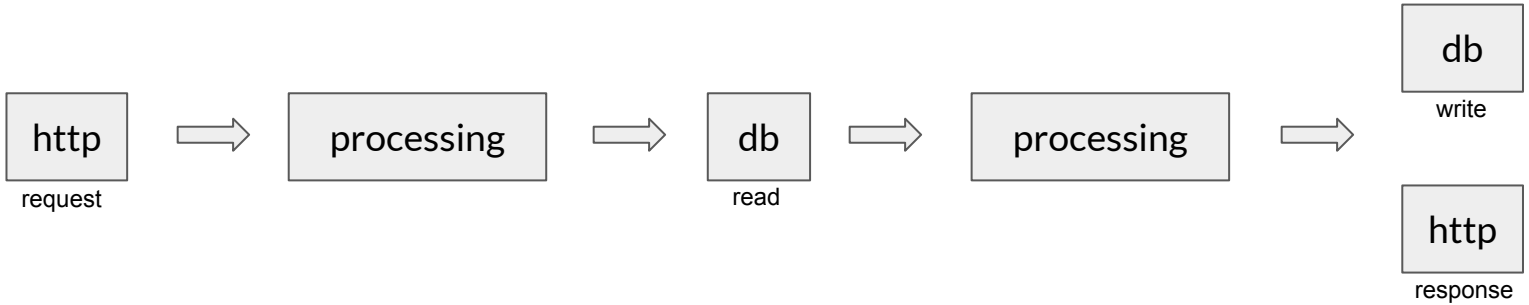
- “Glue”/“Infrastructure” code for unused cases
  - Work around bug that no longer exists?
  - Someone planning for some unused “future”?
- Unused user-facing feature: talk to managers?

Am I working with code that shouldn't exist?

- Often have to reason about multiple layers together
  ⇒ layers maybe better merged / something factored out
- E.g. E2E test page objects

Am I coding for data-flow that never happens?

- Server-side often in response to a single web request with a statically-known data flow



- Be wary of applying patterns more usual to UI where “anything can happen anytime”

Am I more DRY but less explicit or flexible?

- Fewer layers and contracts can be easier to change
- Fewer layers can be easier to reason about

What if I didn’t use a class (hierarchy)?

- Can code that needs state be separated out?
- Can code be re-used through composition?
- If often working with multiple levels of hierarchy when reasoning, hierarchy is probably more hindrance than benefit

It’s a big change and I’m scared: what can I do?

- (Write tests at level covering changes)
- Large (mostly) releasable commit then split, or...
- … baby releasable commits, refactoring first
- Split to other PRs if appropriate
- Tests in the same commit as corresponding change