Today

What is correctness?

How can we know what's right?

What can we do to check our code's correctness?

Correctness

When our code *always* does the right thing

No mistakes!

- "one mistake in a million isn't so bad, is it?"
- processor execute **billions** of instructions per second!
- how about about 1 in 9 billion?

This is a very high bar to clear!

Checking correctness

Ultimately: mathematical *proof*

- possible in some circumstances
- often difficult in practice
- area of active research!

Introduction to the
seL4 proofs

Matthew Brecknell, Data61
@mbrcknl https://m.brck.nl

seL4 Summit 3rd seL4 Summit - November 15-18, 2020

What's the next best thing?

Even some of the highest-assurance systems (e.g., seL4) have a lot of caveats around their proofs of correctness. Their authors really have constructed mathematical proofs of correctness, but as they say, such proofs "prove exactly what you have stated, not necessarily what you mean or what you want". So, while this is a worthwhile activity, and research continues to push new boundaries, most code ________________________________. This is for one of two reasons:

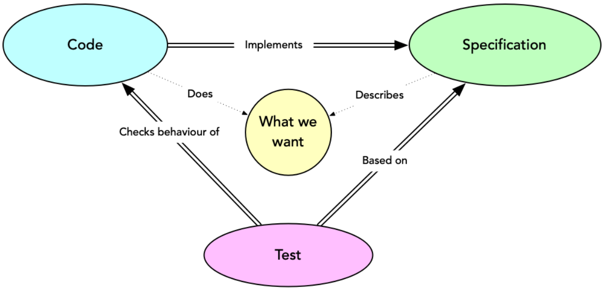
- the code is **beyond the complexity that we can handle proving** or
- the code is **not correct**!

Testing

Code

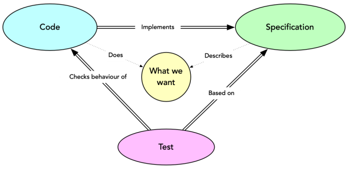
Specifications

Tests



Testing

Code



- what you've worked so hard on
- hopefully does what we want...
- ... but how can we tell?

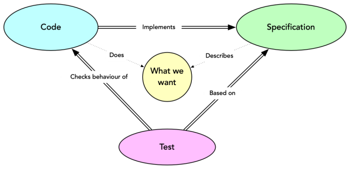
"It works on my computer"

"It works perfectly for me"

Few things are more frustrating than when something doesn't work, you send it in to get fixed and you get the response, "it works for me!" In my last car, the air conditioning often wouldn't turn on. I would bring it into the shop and say, "it's hot outside, the aircon doesn't blow cold air, and my family is really hot all of the time." When the mechanic turned the car on, however, it "worked for them". So was the air conditioning working or not?

Testing

Specification

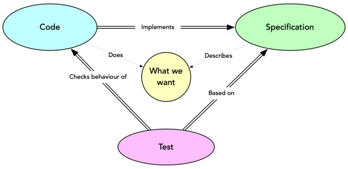


- precise description of requirements
- can never capture everything we *want*
- attempts to describe all *required* behaviour
- like any model, should *correspond* with the real world

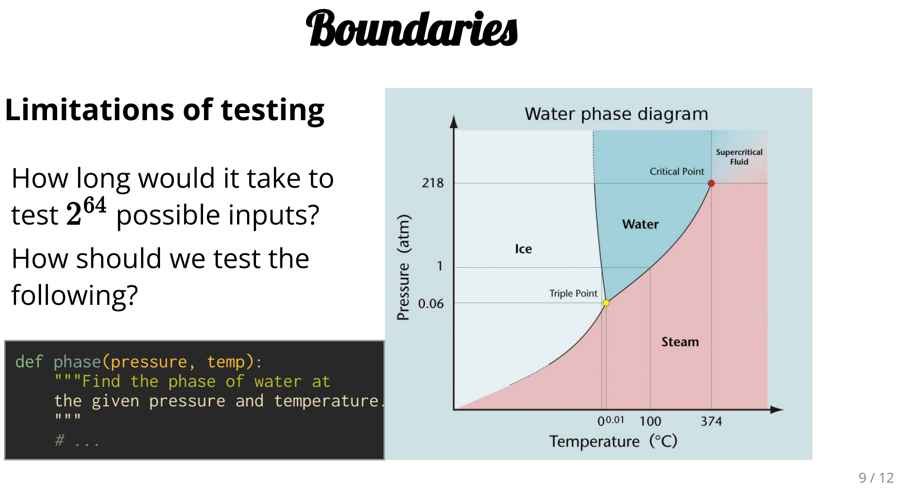
Mathematical models are ______________ and ___________________, which is very helpful for testing. However, if they don't _________________ to the real world, they're not very _____________! It's easy to write a specification that you can live up to where that specification isn't terribly meaningful in the real world... just ask the staff who have to wear winter coats to work in a new building where the HVAC systems have been checked for conformance with the specification!

Testing

Test



- a set of *inputs* and *expected outputs*
- based on the specification
- can be **checked independently** against the specification



Boundaries

Limitations of testing

How long would it take to test 2^{64} possible inputs?

How should we test the following?

```
def phase(pressure, temp):
    """Find the phase of water at
    the given pressure and temperature.
    """
    # ...
```

We usually can't test ____________________, i.e., checking ____________________________ input.

Example*

Given a specific year as an input, a function should return true if the provided year is a leap year and false if it is not.

A year is a leap year if:

- the year is divisible by 4;
- and the year is not divisible by 100;
- unless it is divisible by 400 (when it **is** a leap year)

* From *Software Testing: From Theory to Practice*, Maurício Aniche, 2020. https://sttp.site

Summary

What is correctness?

How can we know what's right?

What can we do to check our code's correctness?