COR 2

Let's talk about edge cases

Say you're writing a function `Node* BST::delete(Node* root, int value)`

- Find the node
- Does it exist?
- Yes
  - No children
  - Left child
  - Right child
  - Left & right children
- And what if the root changes?

Gotta make sure every case works.

Real world complexity management

You're writing more than BSTs most of the time.

- Add new feature to existing pile of code
- Deal with the disk, database, network, OS, third-party code, other services, compilers, *Unicode*, *timezones*, ...
- Comply with internal policies
- Comply with local and foreign laws
- ...

DISCUSSION QUESTION

When in the past were you overwhelmed by complexity?

Type your answer, but wait for our cue to send it.

Test suites: the good, the bad, and the ugly

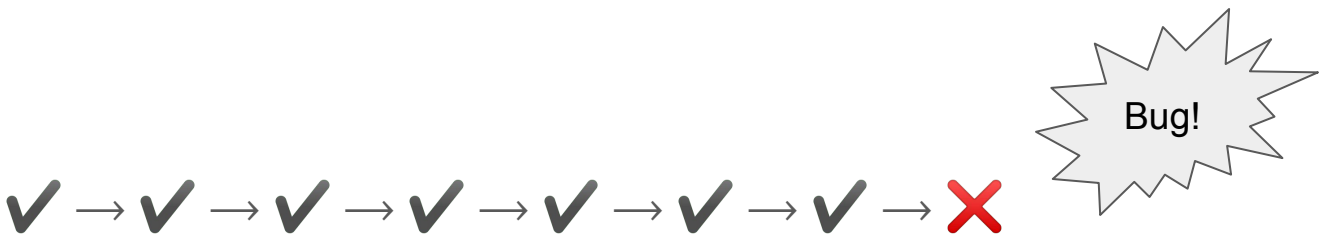
What if I told you... you could codify your specification by checking runtime behavior?

- Write "tests" that exercise your code dynamically
- Check them into your repository
- Run them regularly
- Tests indicate the presence of bugs, but not the absence of them

Invariant of a green main branch

If tests pass on the main branch, the test suite has not surfaced any bugs. It's important that the tests are not flaky.

If, after a change, the the tests no longer pass, it is likely that the change introduced a bug (regression).

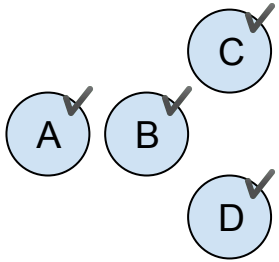


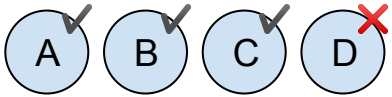
Automating your tests

- Running tests is a pain
- People will avoid it or forget
- Have your code review tool automatically run tests
  - Require them before landing/merging
- "Continuous Integration"—we'll come back to this

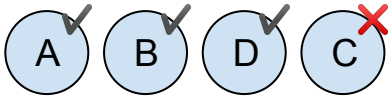
Land races

Land races





or



Could be a new build failure

...or new run-time bug

...or accidentally surface a bug that already existed

...or nothing at all.

The real world, or, pebbles in a stream

- Time passes
- We live in a society
- The world around you changes faster than you can possibly be aware
- Make sure you test in a realistic environment
- Test yourself before you wreck yourself

Anecdote: not being able to send email >500 miles

I cannot do this story justice. Just read it later.

Anecdote: recompiling for different computers & FP

- AVX, AVX-2, AVX-512 are all different vector extensions on Intel/AMD
- AVX vectorizes operations, so AVX vs non-AVX changes math ordering
- Floating point math is not commutative, so errors accumulate
- Probably won't notice this too much unless you do a lot of matrix math

slides JVM

Anecdote: Google CPU failure paper

Google has detected ephemeral computational error in CPUs that are hard to detect and work around.

Law of large numbers at work.

paper

Anecdote: FB hardware failure papers

1. Intermittent hardware (eg RAM) failures cause large-scale performance degradation

2. CPUs have silent data corruption which means your results are just *magically wrong*

paper 1 paper 2

COR 3

How to write useful tests

- Start with the spec
- Write your tests to the spec

...wait, what is a spec?

DISCUSSION QUESTION

What is a specification?

Type your answer, but wait for our cue to send it.

Specification redux

- Describe the software for a target audience, generally:
  - Purpose
  - Interfaces
  - Constraints
  - Assumptions
  - Dependencies
  - Requirements
- Can be formal (IEEE, IETF, W3C, ...) or informal (sketchy Google Doc your coworker threw together while running late to a meeting) or entirely in your head

DISCUSSION QUESTION

Do code comments alone count as a spec?

Type your answer, but wait for our cue to send it.

Comments as a spec

- Can they be turned into documentation (Doxygen, JavaDoc)?
- Do they explain the audience-facing behavior?
- Do they explain *why* a decision was made?

All this to say... it depends. Let's get back to tests.

A new function

```
bool isEven(int num);
```

DISCUSSION QUESTION

How would you test `isEven`?

Type your answer, but wait for our cue to send it.

UTest nuts and bolts

```
TEST(MySoftwareModule, TestName) {
  EXPECT_EQ(some_value, some_expected_value);
}

TEST(MySoftwareModule, SomeOtherTestName) {
  EXPECT_NE(some_value, some_unexpected_value);
}

TEST(AnotherModule, LastTestName) {
  EXPECT_STREQ(some_value, "hello");
}
```

```
[==========] Running 3 test cases.
[ RUN      ] MySoftwareModule.TestName
[       OK ] MySoftwareModule.TestName (631ns)
[ RUN      ] MySoftwareModule.SomeOtherTestName
[       OK ] MySoftwareModule.SomeOtherTestName (631ns)
[ RUN      ] AnotherModule.LastTestName
[       OK ] AnotherModule.LastTestName (631ns)
[==========] 3 test cases ran.
[  PASSED  ] 3 tests.
```

A new function: tests

```
bool isEven(int num);
```

```
TEST(MySoftwareModule, IsEvenWithOddNumberReturnsFalse) {
  EXPECT_EQ(isEven(7), false);
}

TEST(MySoftwareModule, IsEvenWithEvenNumberReturnsTrue) {
  EXPECT_EQ(isEven(8), true);
}
```

A new function: inherently limited tests

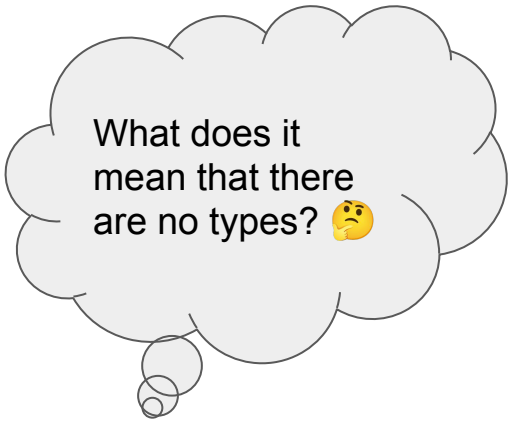
```
bool isEven(int num) {
  switch (num) {
     case 0: case 2: case 4:
     case 6: case 8:
          return true;
     case 1: case 3: case 5:
     case 7: case 9:
          return false;
     // TODO: add the rest of
     // the numbers
     default:
          return false;
  }
}
```

```
TEST(MySoftwareModule, IsEvenWithOddNumberReturnsFalse) {
  EXPECT_EQ(isEven(7), false);
}

TEST(MySoftwareModule, IsEvenWithEvenNumberReturnsTrue) {
  EXPECT_EQ(isEven(8), true);
}
```

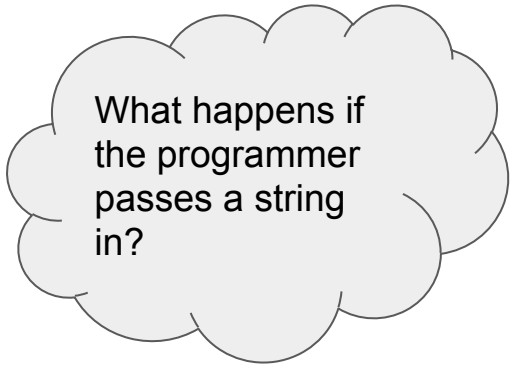
A new function: Python edition

```
def is_even(num):
  if num in (0, 2, 4, 6, 8): return True
  if num in (1, 3, 5, 7, 9): return False
  # TODO: add the rest of the numbers
  return False
```



A new function: Python edition

```
def is_even(num):
  if num in (0, 2, 4, 6, 8): retu    True
  if num in (1, 3, 5, 7, 9): return False
  # TODO: add the rest of the numbers
  return False
```



Testing approaches

Existing code

- Blackbox
- Whitebox
  - Coverage-based

New code

- Test-driven development

Blackbox testing

- Assume you know nothing about the function other than its interface
- Test visible surface of the function

Whitebox testing

- Open the box: what does the code look like?
- Test the tricky-looking bits inside
- Some people go for "coverage"
  - Test suite exercises every line of a function