COR 6: Beyond testing

End-of-semester administrivia

- We updated the Grading section of the syllabus a few weeks ago with concrete information on how your final letter grade will be determined. Please review it if you're curious.
- If you have any concerns or disputes about your grades, the final day to tell us will be **three days after we release HW8 grades**.
- **Remember to fill out your course evaluations!**
    - Due December 15th
    - 7 of you have responded… 11 to go!

This is the last lecture of COR!

We are planning to hold a lecture on Thursday, but it will be **optional** and will cover a topic unrelated to the rest of the course. Some ideas we've had:

- Introduction to using Vim
- Deep dive of an open-source project that uses tools from all four modules
- Tips for ramping up on a big project quickly (places to start, code to read)
- Programming language implementation: interpreters/compilers/machine code
- Introduction to debugging practices & philosophies
- Introduction to performance and profiling

If any of these catch your eye, or if you have a different idea, see the next slide!

DISCUSSION QUESTION

What do you want to learn on Thursday?

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

Limitations of tests

Tests only make guarantees about certain exposed runtime behavior of programs.

Verification: “Are we building the product right?” **Validation**: “Are we building the right product?”

—Boehm

Coding practices

See: MISRA C, JPL C, ...

Code review

Other ways to check runtime behavior

- Dynamic invariant checks inside software (“asserts”)
  - Including compiler instrumented dynamic invariant checks (ASAN, UBSAN, ...)
- Emulation (Valgrind)
- Fuzzing/random testing
  - AFL/AFL++
- Logging
- Property-based testing
  - QuickCheck/LeanCheck (Haskell)
  - Hypothesis (Python)
- Bounded exhaustive testing
- Characterization tests
  - Good for images, large outputs, legacy codebases, etc
- User bug reports / changes in metrics

asserts

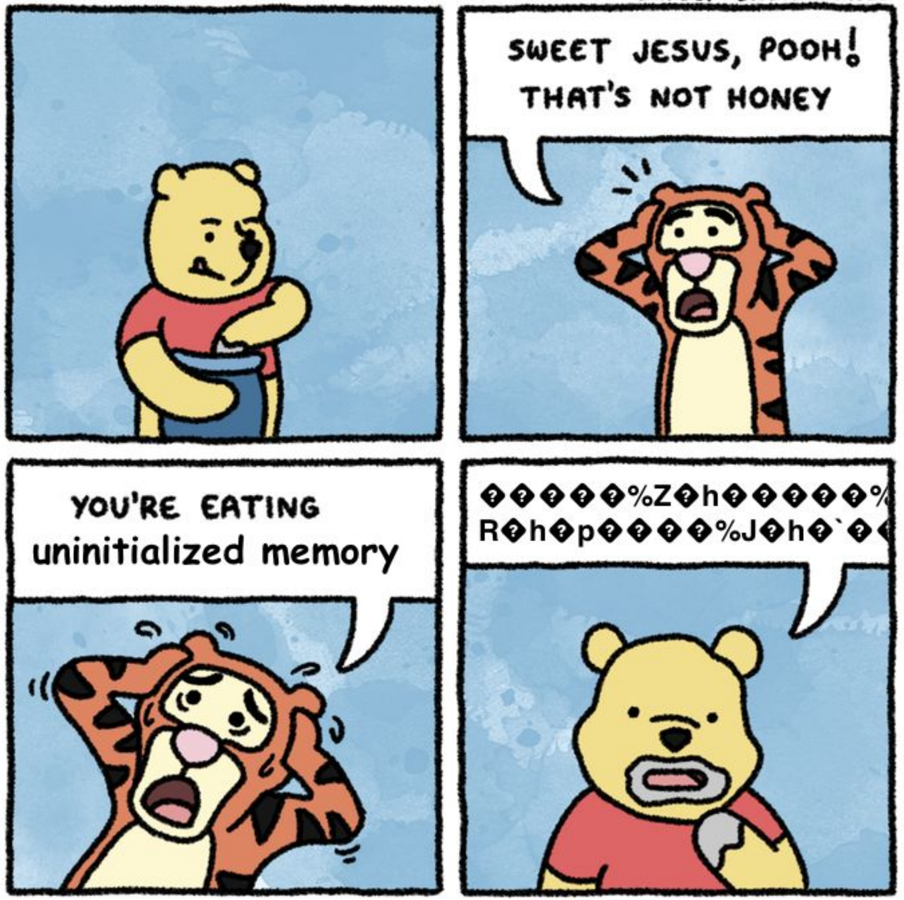
```
#ifndef NDEBUG
#define assert(condition) if (!(condition)) abort()
#else
#define assert(_condition)
#endif

bool mod(int left, int right) {
  assert(left >= 0);
  assert(right > 0);
  return left % right;
}
```

ASAN/UBSAN

- ASAN: AddressSanitizer
- UBSAN: UndefinedBehaviorSanitizer
- And others, too

- Compiler is modified to run extra code for every allocation, free, and pointer dereference.
- Keeps track of which memory is allocated.
- Accesses that touch unallocated memory immediately abort the program.

borrowed from SAFELY ENDANGERED


SWEET JESUS, POOH! THAT'S NOT HONEY
YOU'RE EATING uninitialized memory
�����%Z�h������%
R�h�p�����%J�h�`��

Valgrind

- Run code in a sandbox
- Run small snippets of code before and after memory read/write
- Mark which memory is allocated and which is freed
- Slower than ASAN but more precise

Fuzzing

- Generate random inputs to code to see what breaks
- Often reveals unsafe code because the input is completely outside the realm of "normal"
- Some fuzzers, like AFL, instrument the code (like ASAN does) so they can tell what code paths get run for a given input. Then they generate inputs to try and exercise every code path.

- Don't be annoying and file fuzz reports to open source projects without offering a fix.

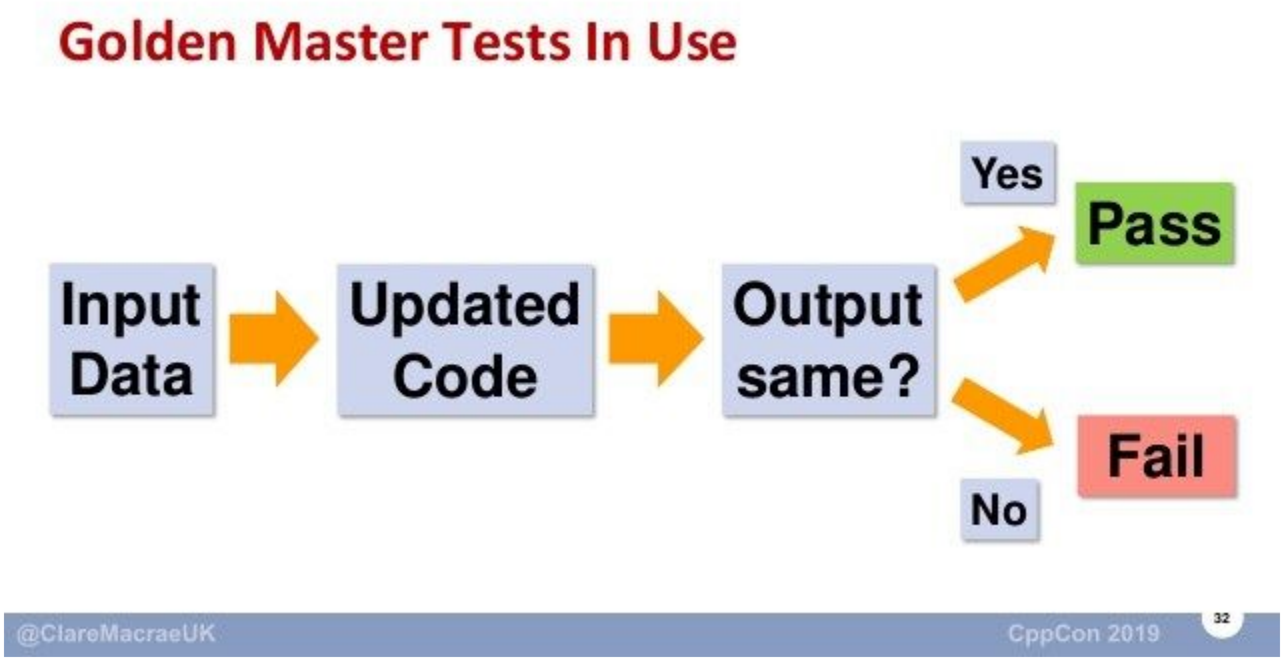
Property-based testing (C++, Python, Haskell, ...)

```
#include <rapidcheck.h>
#include <vector>
#include <algorithm>

int main() {
  rc::check("double reversal yields the original value",
            [](const std::vector<int> &l0) {
              auto l1 = l0;
              std::reverse(begin(l1), end(l1));
              std::reverse(begin(l1), end(l1));
              RC_ASSERT(l0 == l1);
            });

  return 0;
}
```

This slide deck is a good read

Characterization tests/approval tests

- "lock down" current behavior



Golden Master Tests In Use

Input Data

Updated Code

Output same?

Yes

Pass

No

Fail

@ClareMacraeUK

CppCon 2019

32

Thoughts on Golden Master Tests

- Good to start testing legacy systems
- Good when goal is to keep behaviour unchanged
- Depends on ease of:
  - Capturing output
  - Getting stable output
  - Reviewing any differences
  - Avoiding overwriting Golden Master by mistake!

Logging in production

- Don't log PII
- Log *useful* pieces of information
  - Error happened
  - …
- Beware of log spam!
  - Easy to log too much noise and hide the signal
  - One solution: runtime-togglable log messages (by subsystem, severity, etc)
  - Good: Linux kernel dynamic debug framework
  - Bad: Android logcat

Metrics and user bug reports

- Keep an eye on your core metrics
  - Including number of bugs reported
- When they dip unexpectedly (not just "nighttime"), something might have happened

Ideas:

- Time spent in app
- Distribution of exit codes

What constitutes "behavior"?

- Function results in memory
  - Easy to test
- I/O actions (syscalls), including their order
  - Hard to test
- **Performance**
  - ???

The parable of the intern

- Team writes a specification for a sorting function
- Specification describes the domain (integers), that it is a stable sort, etc
- Lead engineer writes the most beautiful merge sort you have ever seen
  - Writes a bunch of tests too
- Code review looks good, so the code lands
- Intern discovers a weird bug and decides to just rewrite as an insertion sort because it's simpler
- Mentor says "that's probably fine; we don't sort that many numbers anyway" and accepts the diff
- All tests pass, so the code lands
- ????
- Benchmarks

Ways to check behavior statically

- Type systems
- Static analysis
  - Symbolic execution / taint/dataflow analysis
- Proofs (Coq, Isabelle, ...)
- Model checking (Alloy, TLA+, ...)

A new function: Python edition

```
def is_even(num):
  return num % 2 == 0
```

DISCUSS: What does it mean that there are no types? 🤔

A new function: C edition

```
bool isEven(int num) {
  return num % 2 == 0;
}
```

More types

```
Result<FILE*, Error> openFile(const char* filename);
```

```
Option<int> List::find(T value);
```

- Stripe, Instagram, others are deciding that they cannot develop large applications without a static type checker

Rust, OCaml, Haskell, Idris, Agda, and more

- Ownership
- Thread safety
- Dependent types
- ...

Static analysis

```
int average(vector<int> nums) {
  return sum(nums)/nums.size();
}
```

DISCUSS:
What's wrong
with this code?

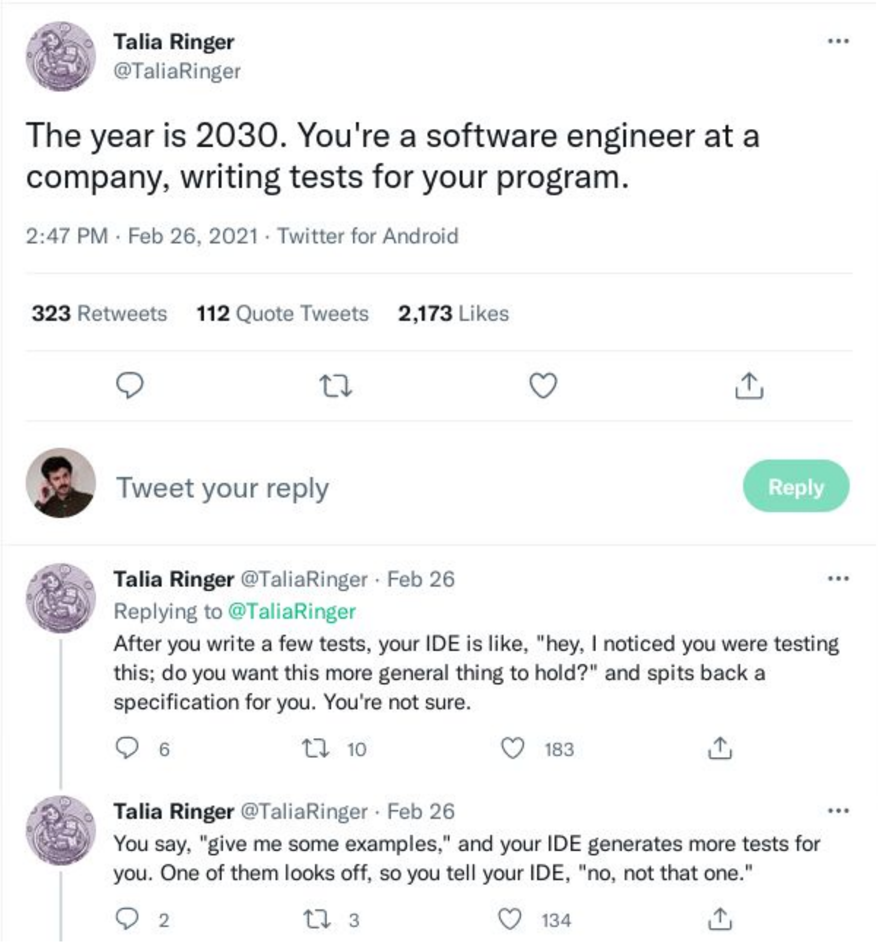
Static analysis

```
int average(vector<int> nums) {
  size_t num_items = nums.size();  // num_items can be 0 here
  if (num_items == 0) {
    // num_items is definitely 0 here
    abort();
    // or do something else defined by your application
  }
  return sum(nums)/num_items;  // now num_items can't be 0 :)
}
```

Proofs: what problems does Coq solve?

Model checking: what problems does TLA+ solve?

The future of proof engineering



Talia Ringer
@TaliaRinger

The year is 2030. You're a software engineer at a company, writing tests for your program.

2:47 PM · Feb 26, 2021 · Twitter for Android

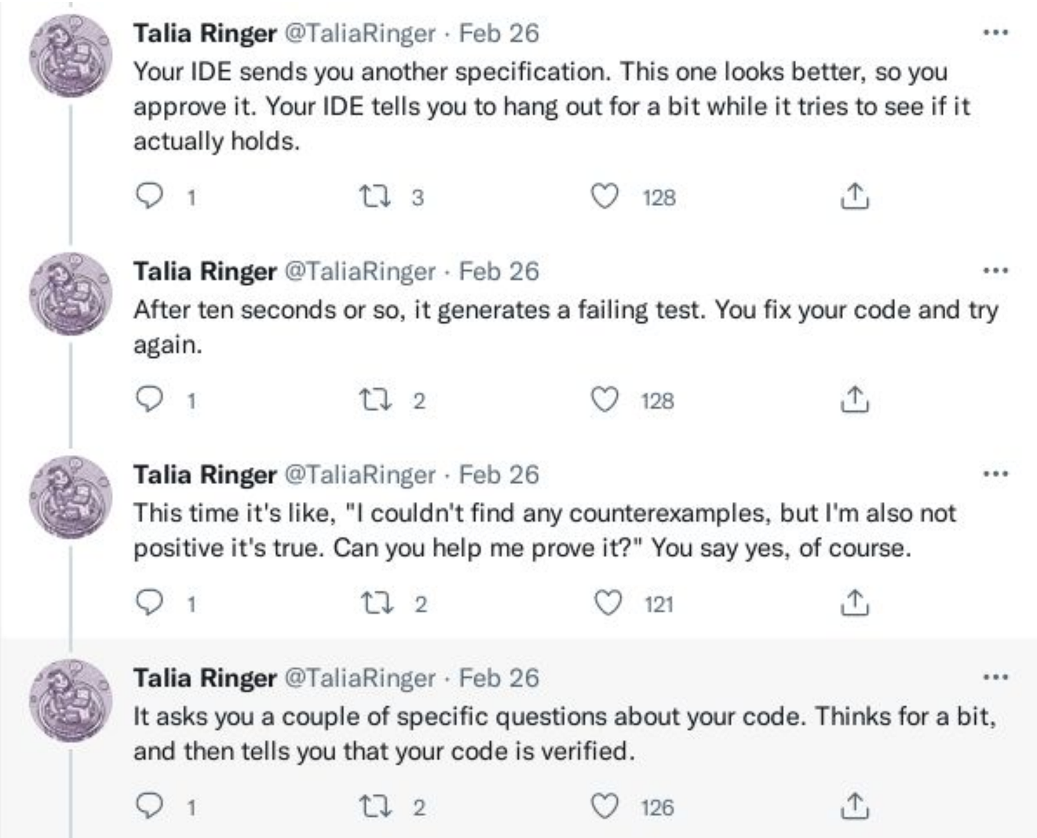
323 Retweets **112** Quote Tweets **2,173** Likes

Talia Ringer @TaliaRinger · Feb 26
Replying to @TaliaRinger
After you write a few tests, your IDE is like, "hey, I noticed you were testing this; do you want this more general thing to hold?" and spits back a specification for you. You're not sure.

6 10 183

Talia Ringer @TaliaRinger · Feb 26
You say, "give me some examples," and your IDE generates more tests for you. One of them looks off, so you tell your IDE, "no, not that one."

2 3 134



Talia Ringer @TaliaRinger · Feb 26
Your IDE sends you another specification. This one looks better, so you approve it. Your IDE tells you to hang out for a bit while it tries to see if it actually holds.

1 3 128

Talia Ringer @TaliaRinger · Feb 26
After ten seconds or so, it generates a failing test. You fix your code and try again.

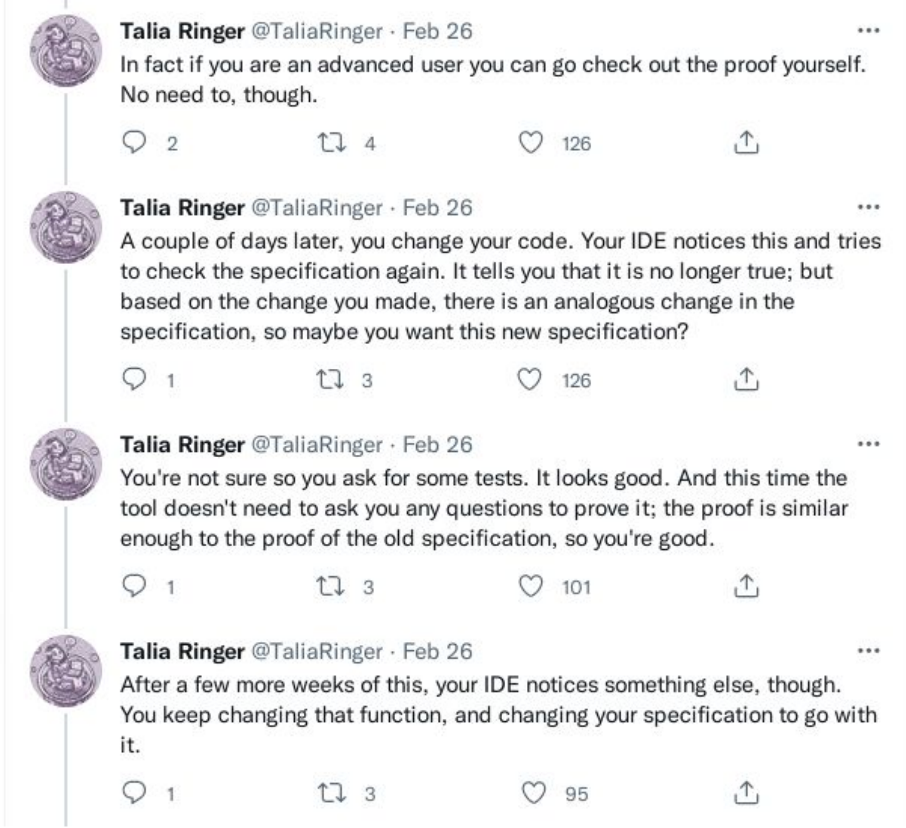
1 2 128

Talia Ringer @TaliaRinger · Feb 26
This time it's like, "I couldn't find any counterexamples, but I'm also not positive it's true. Can you help me prove it?" You say yes, of course.

1 2 121

Talia Ringer @TaliaRinger · Feb 26
It asks you a couple of specific questions about your code. Thinks for a bit, and then tells you that your code is verified.

1 2 126



Talia Ringer @TaliaRinger · Feb 26 ···

In fact if you are an advanced user you can go check out the proof yourself. No need to, though.

💬 2 🔁 4 ♡ 126

Talia Ringer @TaliaRinger · Feb 26 ···

A couple of days later, you change your code. Your IDE notices this and tries to check the specification again. It tells you that it is no longer true; but based on the change you made, there is an analogous change in the specification, so maybe you want this new specification?

💬 1 🔁 3 ♡ 126

Talia Ringer @TaliaRinger · Feb 26 ···

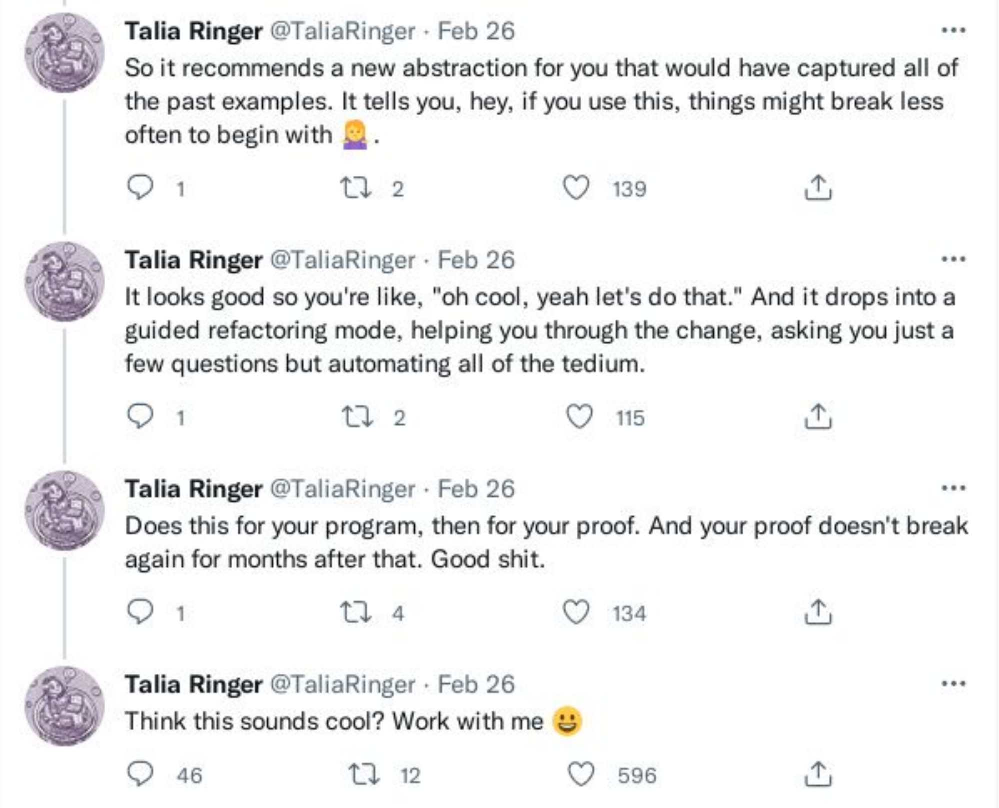
You're not sure so you ask for some tests. It looks good. And this time the tool doesn't need to ask you any questions to prove it; the proof is similar enough to the proof of the old specification, so you're good.

💬 1 🔁 3 ♡ 101

Talia Ringer @TaliaRinger · Feb 26 ···

After a few more weeks of this, your IDE notices something else, though. You keep changing that function, and changing your specification to go with it.

💬 1 🔁 3 ♡ 95



Talia Ringer @TaliaRinger · Feb 26 ···

So it recommends a new abstraction for you that would have captured all of the past examples. It tells you, hey, if you use this, things might break less often to begin with 🤷‍♀️.

💬 1 🔁 2 ♡ 139

Talia Ringer @TaliaRinger · Feb 26 ···

It looks good so you're like, "oh cool, yeah let's do that." And it drops into a guided refactoring mode, helping you through the change, asking you just a few questions but automating all of the tedium.

💬 1 🔁 2 ♡ 115

Talia Ringer @TaliaRinger · Feb 26 ···

Does this for your program, then for your proof. And your proof doesn't break again for months after that. Good shit.

💬 1 🔁 4 ♡ 134

Talia Ringer @TaliaRinger · Feb 26 ···

Think this sounds cool? Work with me 😃

💬 46 🔁 12 ♡ 596

Don't forget to submit your course eval!