test-first development

So... write the tests first, then the program!?

Sounds backward, but...

This was the #1 request from UNI alumni when asked for testing topics that should be covered.
(alert: the audience was biased, not a full survey)

What does it look like?

It can look like many things. Here is an example
— not quite the loosest possible, where you write tests and then use them as a guide
— not quite the tightest possible, which I'll describe later.

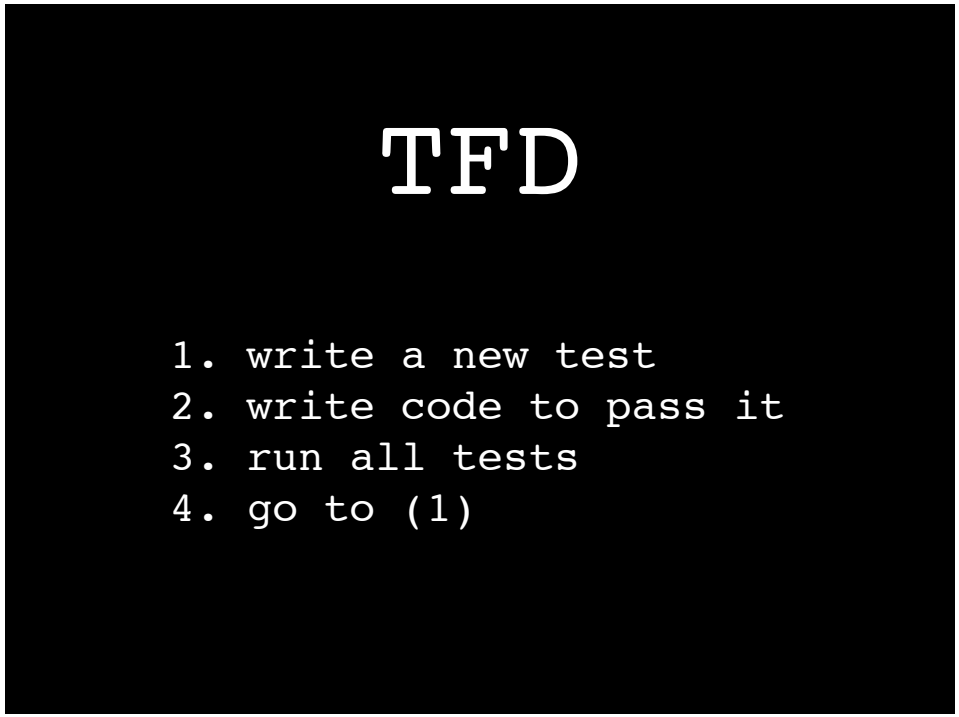
reconsider...

```
Tests = [
    # String  Prefix  Expected
    ['a',     'a',    True],
    ['a',     'b',    False],
    ['abc',   'a',    True],
    ['abc',   'ab',   True],
    ['abc',   'abc',  True],
    ['abc',   'abcd', False],
    ['abc',   '',     True]
]
```

Un-comment one at a time, broadening the implementation as needed:

```
return true
return str = prefix
return str[0:1] = prefix
return str[0:len(prefix)] = prefix
```

Are we there yet?

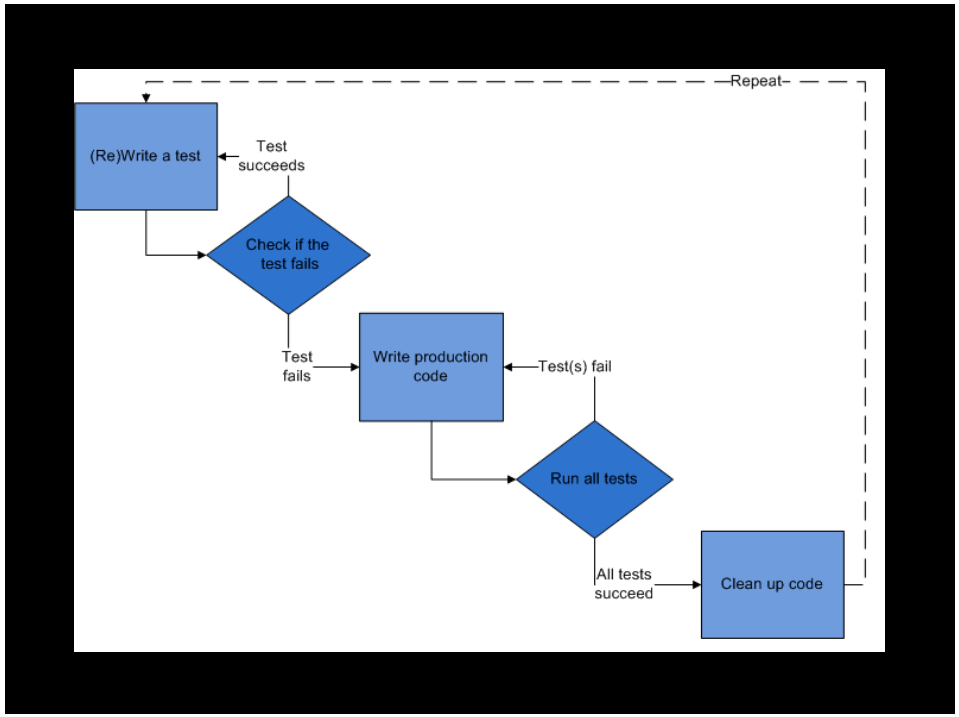


In the most extreme form of TDD, found among many in extreme programming, the developer would implement her code in steps so small that each version handles **exactly one more** test case!

“red bar – green bar – refactor

1. ... that fails
2. ... just enough to make it pass

What's the difference test-**first** development test-**driven** development?



We refactor after the test pass.

TDD in XP

1. write a new test
2. write code to pass it
3. run all tests
4. **refactor**
5. go to (1)

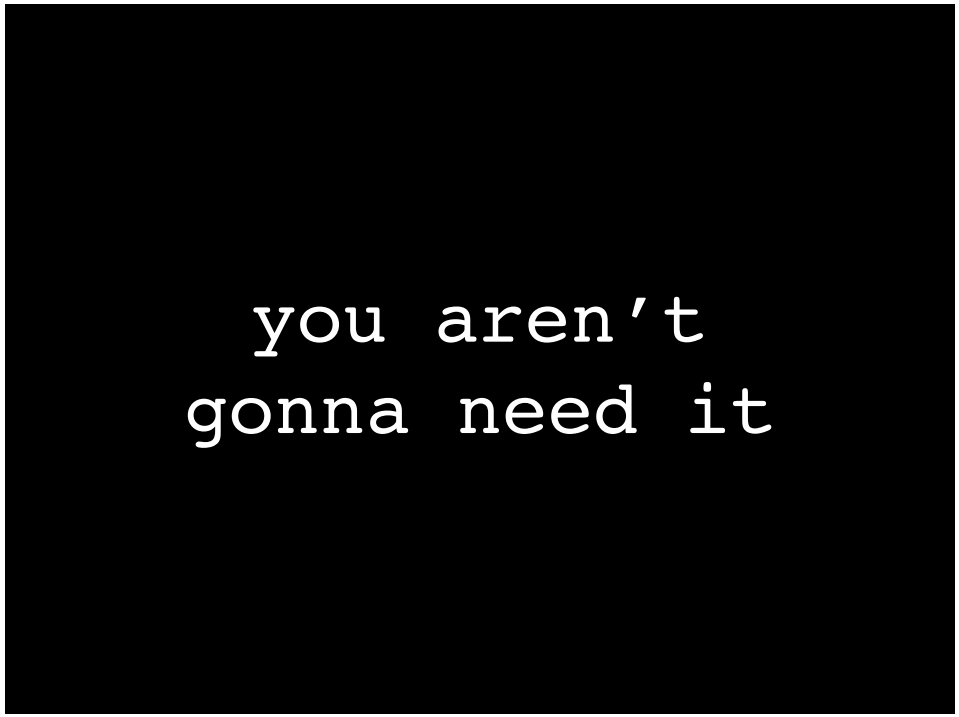
Why would we have to refactor every time?

In the most extreme form of TDD, found among many in extreme programming, the developer would implement her code in steps so small that each version handles **exactly one more** test case!

“red bar – green bar – refactor”

1. ... that fails
2. ... just enough to make it pass

do the
simplest thing
that could
possibly work


you aren’t
gonna need it

Simple design with refactoring can generate simple, clean complex designs.

Tests are essential to effective refactoring.

If you are going to do this all the time — literally — you need convenience in writing tests and good feedback from the testing framework.

JUnit and its progeny

SUnit from Smalltalk begat JUnit, which was written by Kent Beck and Erich Gamma in 1997. JUnit made testing, especially unit testing, easy enough that programmers **actually started doing it**. It is now integrated into most Java IDEs.

JUnit begat a whole family of xUnit implementations. Work-alikes are available for C++, .NET, Ruby, Perl, Scheme, and even Ada. Many are now built-in to the languages, or come as standard packages. `unittest` comes as standard package in Python.

Once you know one of these tools, you can easily learn and use the others.

Agile developers have created add-ons for measuring test execution times, recording tests, testing web applications, and many other extensions.

```
class TestAddition(unittest.TestCase):

    def test_zeroes(self):
        self.assertEqual(0 + 0, 0)
        self.assertEqual(5 + 0, 5)
        self.assertEqual(0 + 13.2, 13.2)

    def test_positive(self):
        self.assertEqual(123 + 456, 579)
        self.assertEqual(1.2e20 + 3.4e20, 3.5e20)

    def test_mixed(self):
        self.assertEqual(-19 + 20, 1)
        self.assertEqual(999 + -1, 998)
        self.assertEqual(-300.1 + -400.2, -700.
```

A simple example in Python's `unittest`.

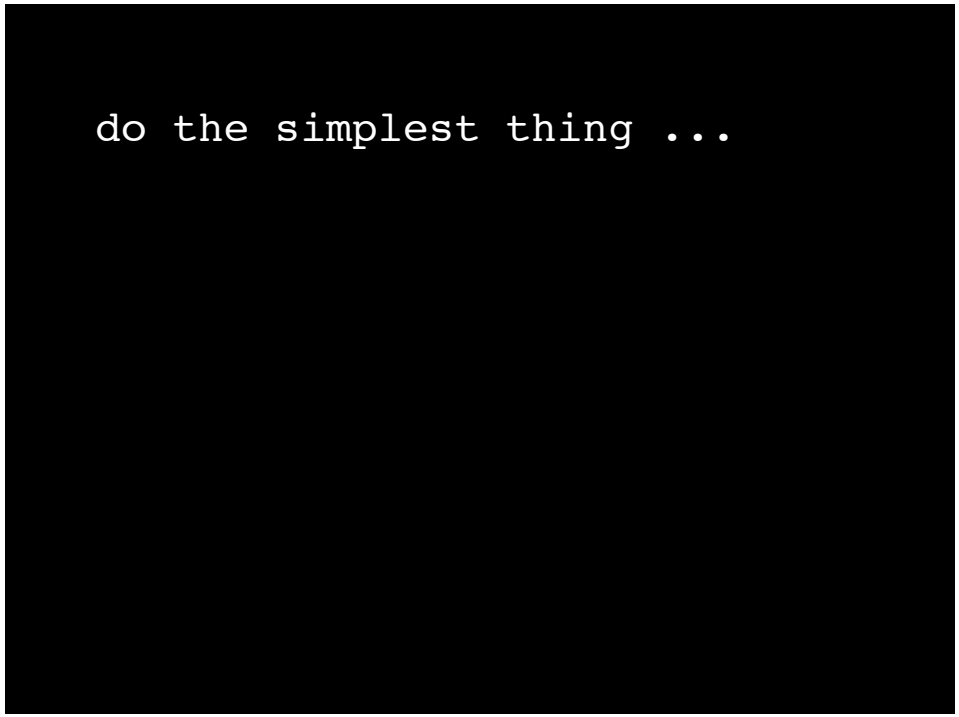
```
> python test-addition.py
.F.
[snip]
```

[... run it ...]

The error message gives useful feedback on the failure.
When all tests pass, the output is clean.
[... fix it / run it ...]

Many developers use graphic interfaces to xUnit, and actually see red bars and green bars.

[... look at `unittest` implementation of our `starts_with` test ...]


do the simplest thing ...

do the simplest thing ...

write tests firsts

do the simplest thing ...

write tests firsts

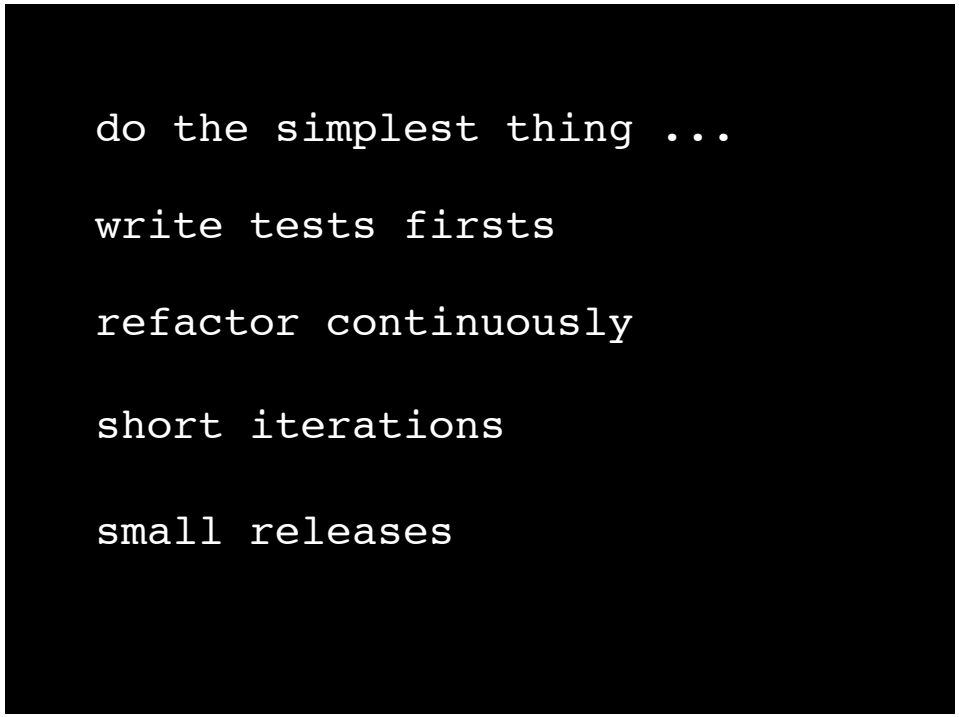
refactor continuously

do the simplest thing ...

write tests firsts

refactor continuously

short iterations



... plus a few other practices:

- **pair programming**
- collective code ownership
- sustainable pace

... consists of this set of development practices. It is the prototypical **agile** approach to software development.

If we want to boil XP down to one phrase, this it.

It works pretty well to summarize most agile approaches.

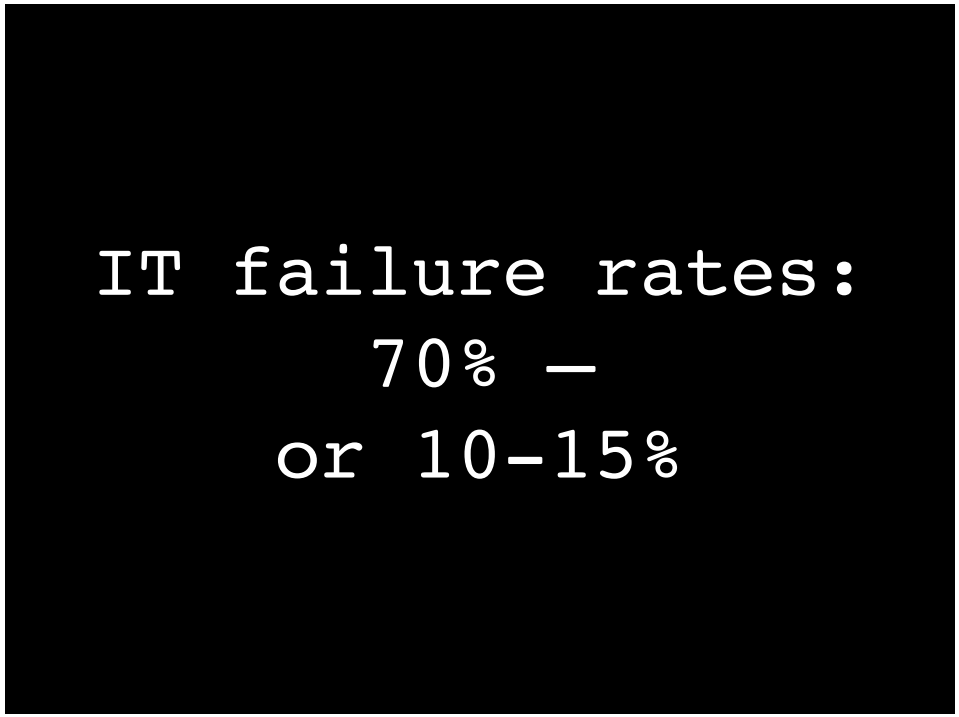
metaphor redux

The semester is over.

You now know something about software engineering (I hope): what it is, what it means, what we do.

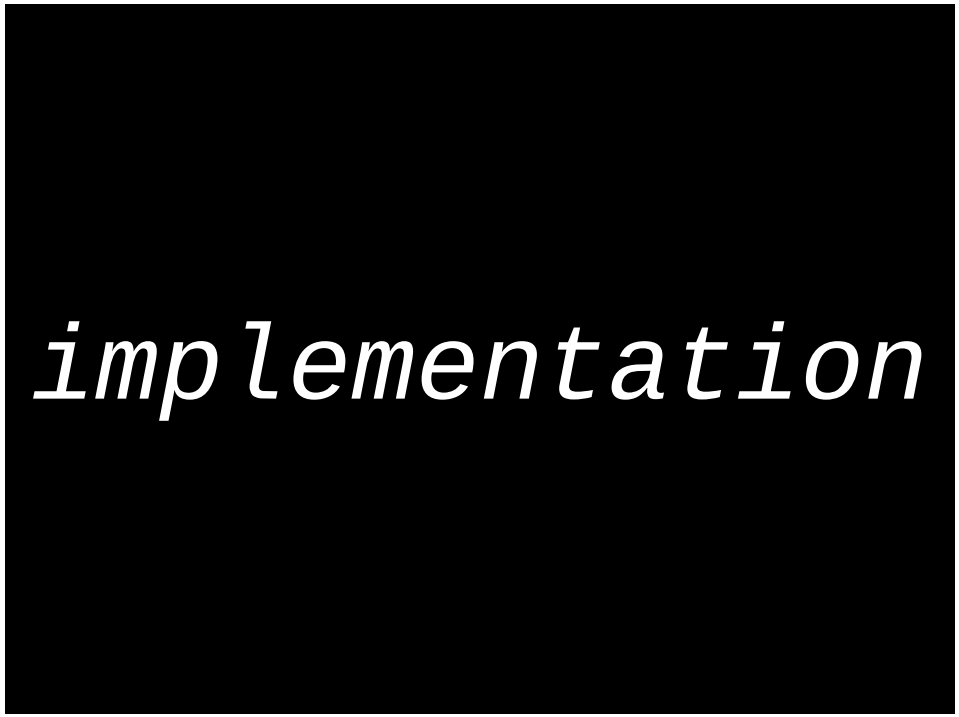
Let's step back and look at the origin of the metaphor, and how it helps — and hinders — us.

- exercises one component in isolation condition
- works from the developers perspective, by testing the program's internals

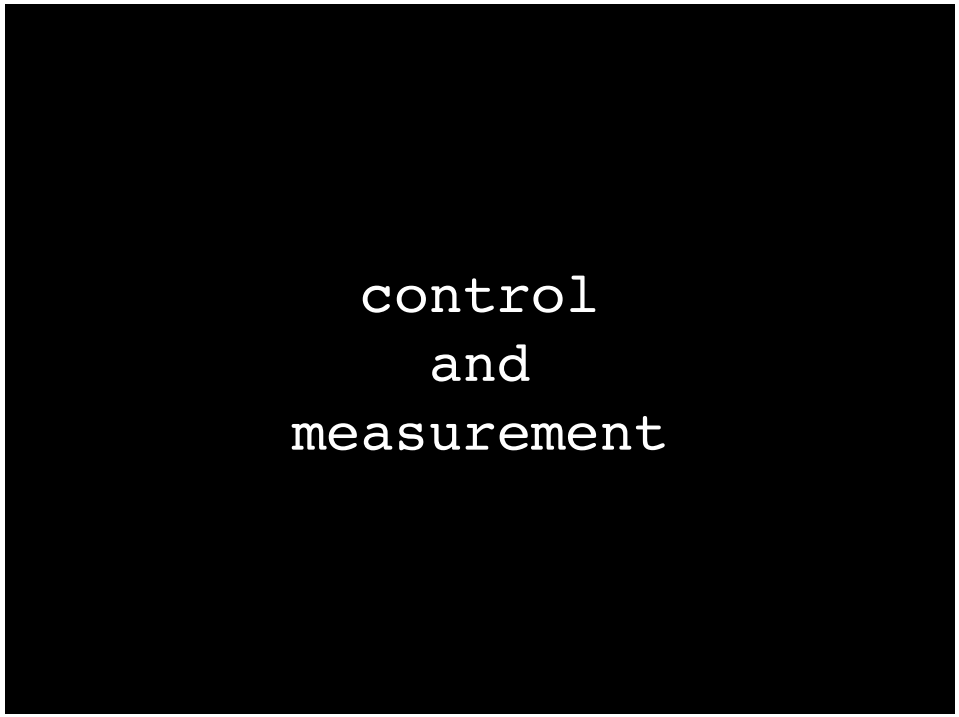


One motivation for the **need for** software engineering was the “software crisis”.
Is there one? If so, how bad is it?

... data from the 1994 Standish Report.
The data is not open, nor is the methodology.
Many have questioned the scale and severity of the so-called crisis, even from the beginning.
Glass’s article is from 2005.



DeMarco writes in this space. He was one of the creators of the discipline, not a long-time skeptic.



The goal is repeatability. This ultimately means learning. Good!

There is more...

Project A will eventually cost about a million dollars and produce value of around $1.1 million.
Project B will eventually cost about a million dollars and produce value of more than $50 million.

What are the trade-offs?

Does control work? Do people actually follow the script?

... an idea whose time has come and gone. I still believe it makes excellent sense to engineer software. But that isn't exactly what software engineering has come to mean.

We should...

learn from engineering

Everything we have discussed this semester is important. It is all a part of how we make software. If we want to get better, then we have to **pay attention** and **feed results back into the process**.

At its simplest, this includes:
- measure where possible and cost-effective
- use the data to improve our process

do not limit ourselves (unnecessarily)

What makes sense as “engineering your software” depends on the **context**: the size of the organization, the kind of software, ...

finis

Questions?

Final exam period:
- exam
- project follow-up (presentation, team evals, debrief)

Do you care which order?

** **There is a reading assignment for the final!!** **