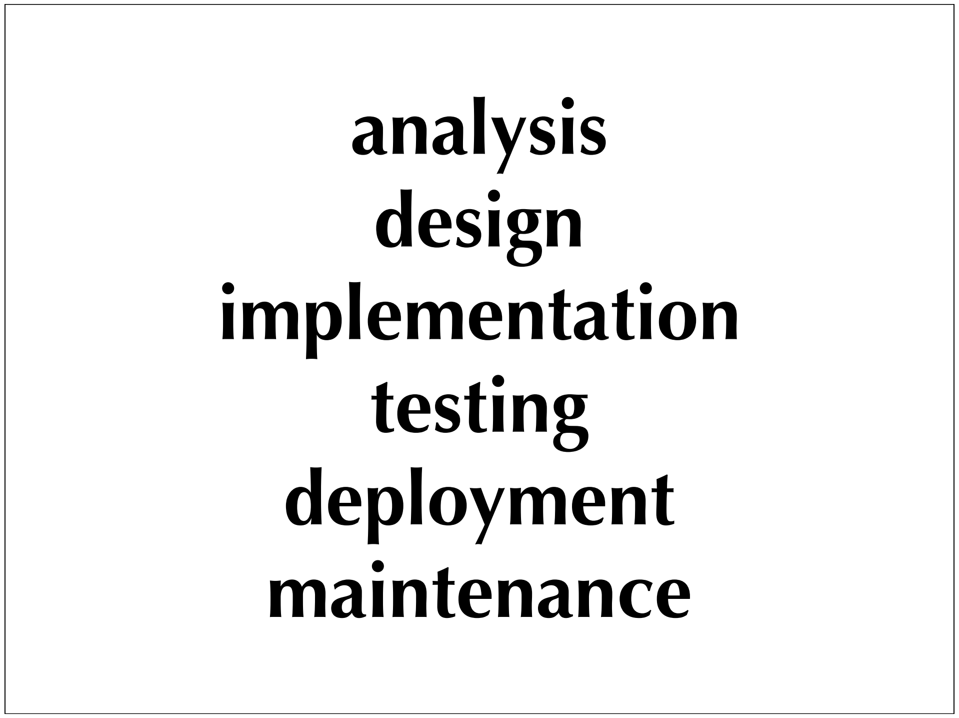
Identify **the nastiest bug** you have ever faced, in a class assignment or in an outside job.

Identify how you found and fixed it.

List all the ways you could have found and/or fixed it.

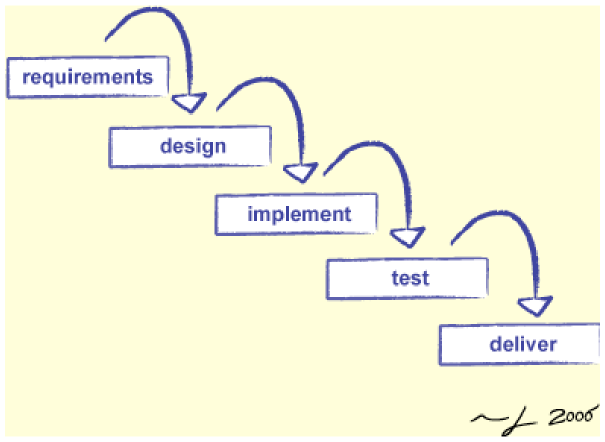
... Get in groups to share answers and discuss.

Tools? Process?
Discipline.

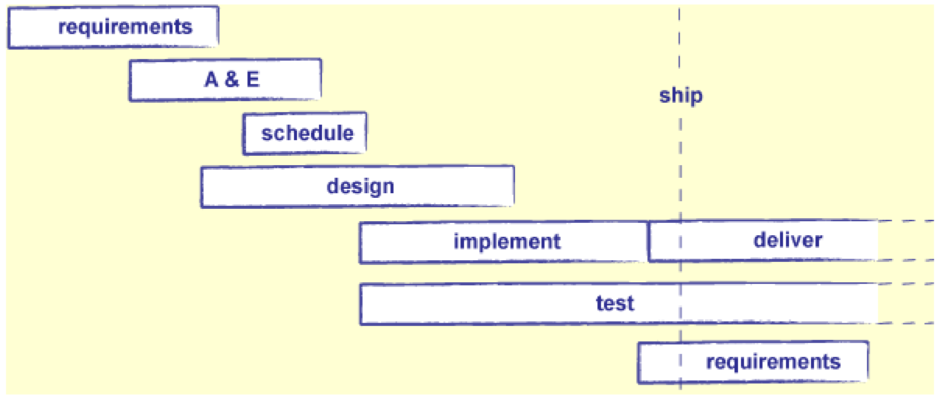


analysis
design
implementation
testing
deployment
maintenance

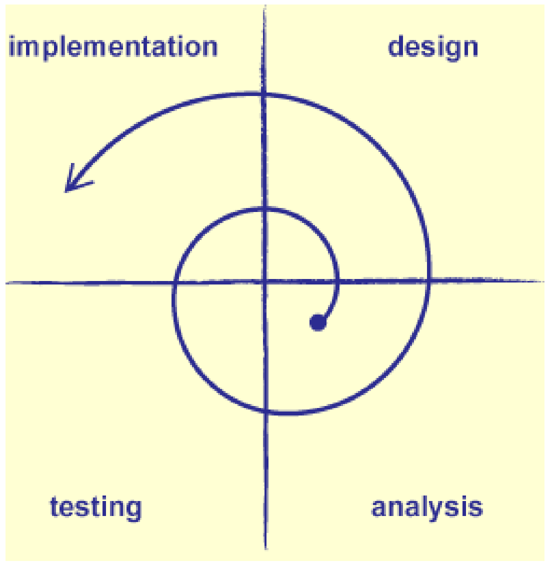
These are stages in the **life** of software.
They can also be stages in the **making** software.



Waterfall Model -- a natural approach, especially when drawing from engineering.
Do real projects go this way? Perhaps still useful as the model.
Problem: lack of accountability.

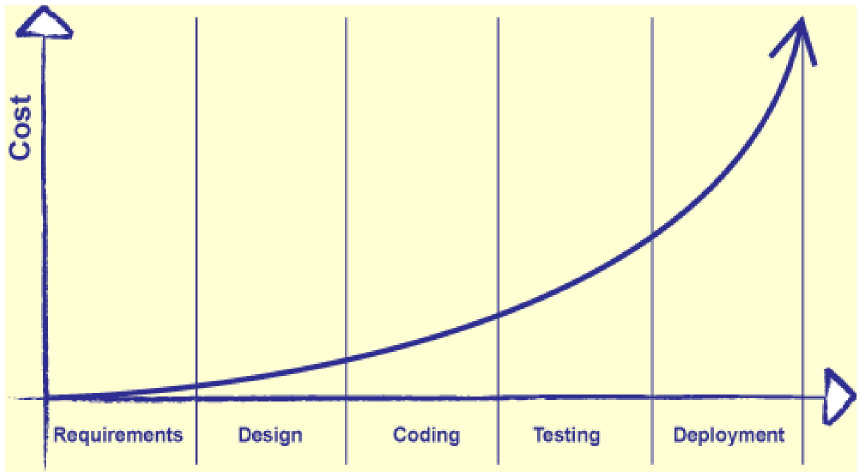


An alternative that recognizes that phases overlap and circle back around.



An alternative that builds on the idea that phases **do and must** circle back around.

The agile methods — XP, Scrum, Crystal Light, ... — all build on this.
How quickly do you spiral around?



Boehm's Curve. The idea: Change costs more the farther you go in the product lifecycle.

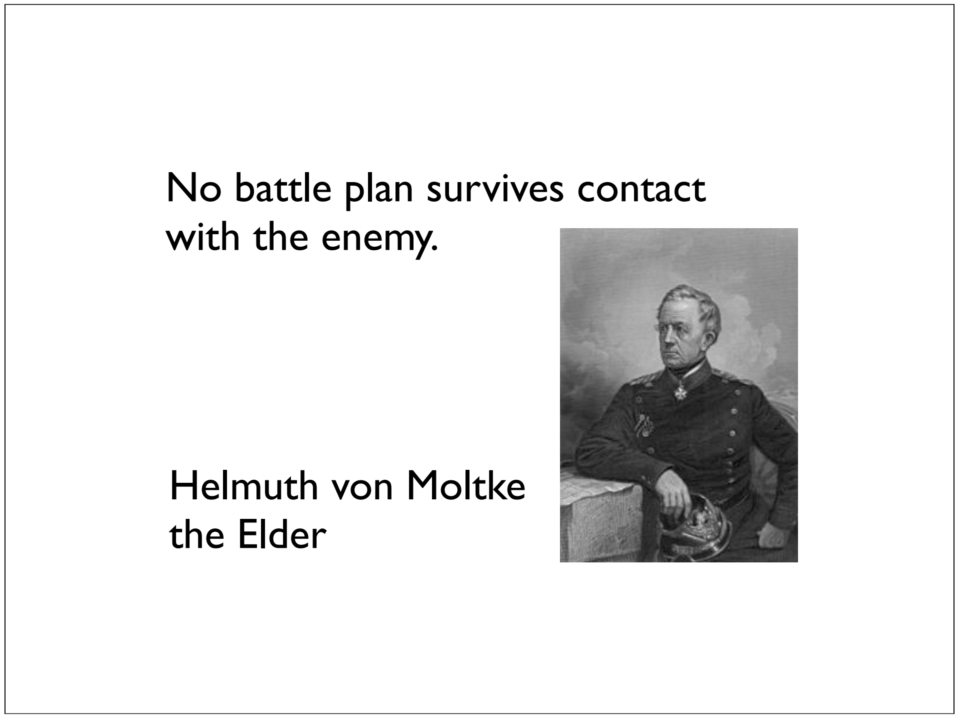
(But how much? And can we change the shape of the curve?)

“Sturdy” Processes

measure twice, cut once

Think through users' needs, design, and possible problems before starting to code

Inspired by traditional engineering, in which late changes are **very** expensive.

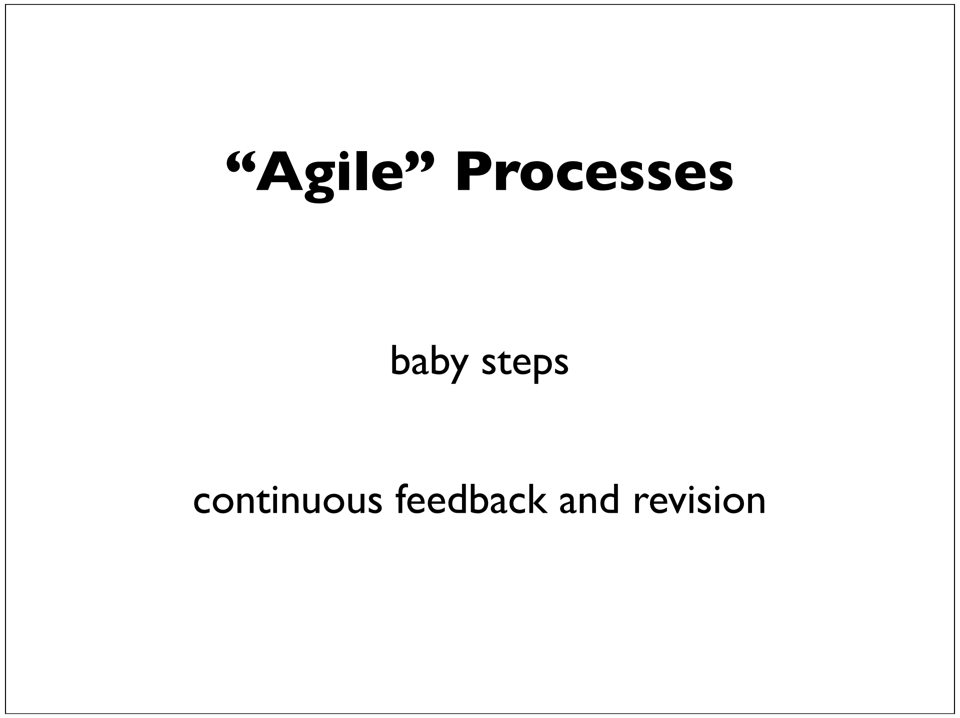


No battle plan survives contact with the enemy.

Helmuth von Moltke the Elder

Respond to the limits felt within sturdy approaches...
Recognize reality.

... or giving up hope of doing better? **(question)**

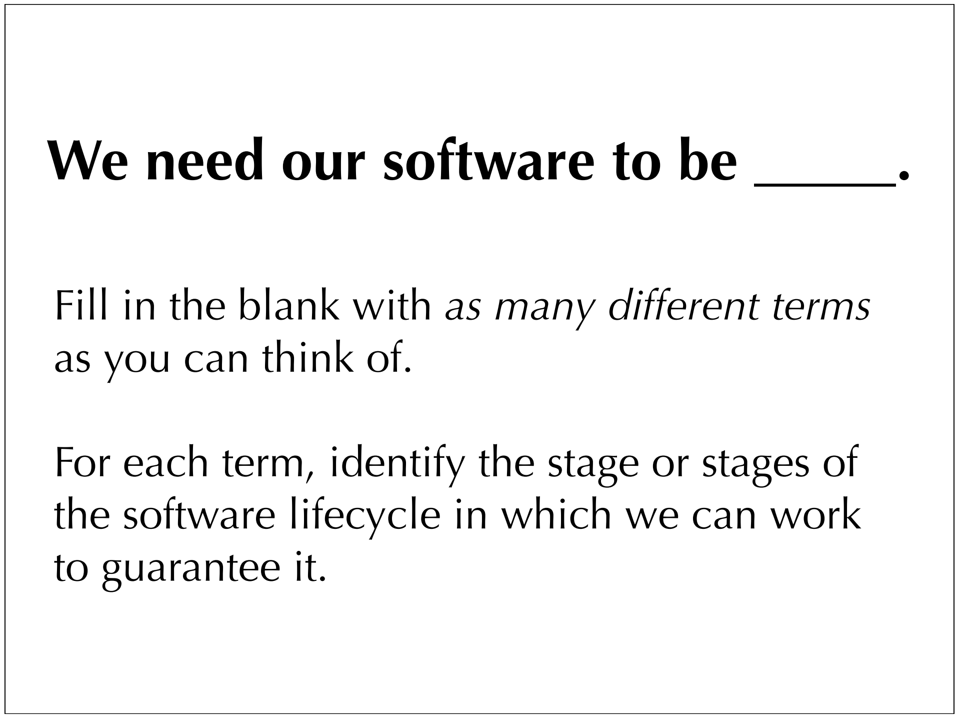

“Agile” Processes

baby steps

continuous feedback and revision

Does change cost less with software?
Question: How could it?

Can we change the shape of the curve — with tools or process?



We need our software to be _____.

Fill in the blank with *as many different terms* as you can think of.

For each term, identify the stage or stages of the software lifecycle in which we can work to guarantee it.

How can we know that our software is _____?
How do we ensure it?

Trade-offs: among goals, among choices for any goal.

These are the things we must think about when we make software.

These are the things that makes software development challenging.

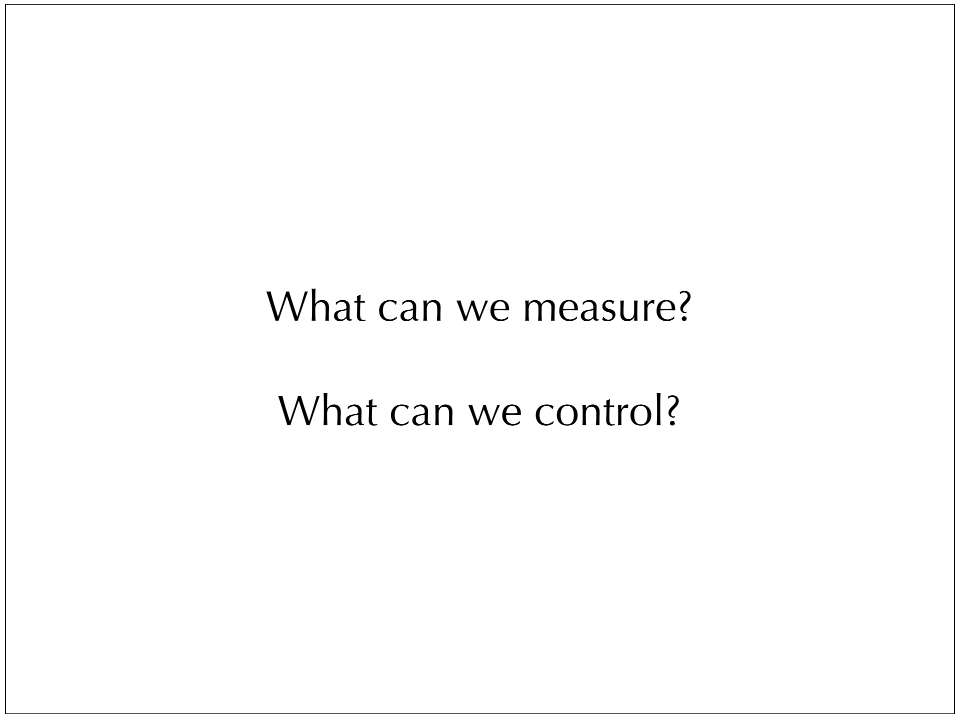
How can we tell?

metrics

empirical software engineering

software metrics
process metrics

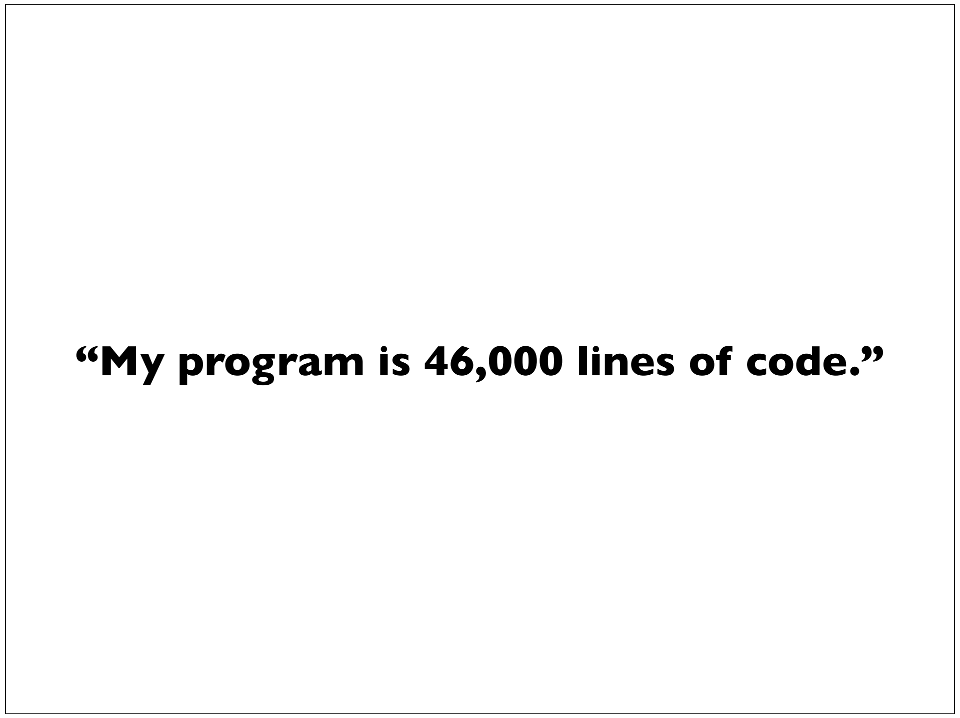
Empirical SE is one of the great scientific opportunities we have. It’s hard.
Empirical Studies of Programmers.



What can we measure?

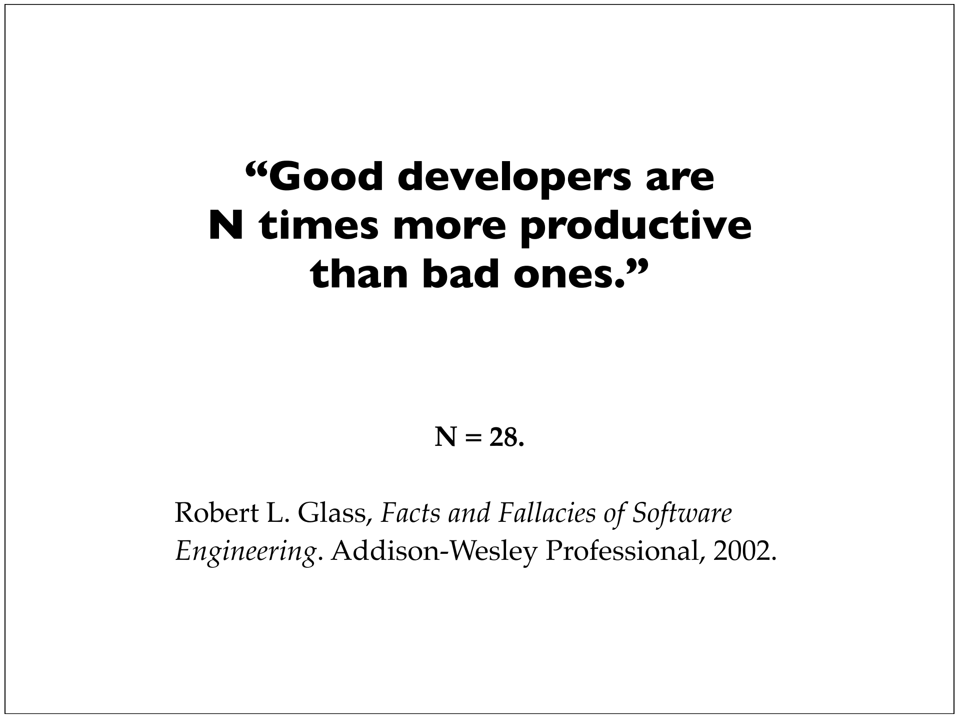
What can we control?

Is that what we want to measure?
How much control do we want?



“My program is 46,000 lines of code.”

LOC is a great example of an **ineffective metric**: not want you want to control, measure, reward.
But what?   Function points, ...



“Good developers are N times more productive than bad ones.”

N = 28.

Robert L. Glass, *Facts and Fallacies of Software Engineering*. Addison-Wesley Professional, 2002.

A **claim with evidence.**
You will see different values of N, but almost always > 10.

(This is one reason you want to be a good developer.)

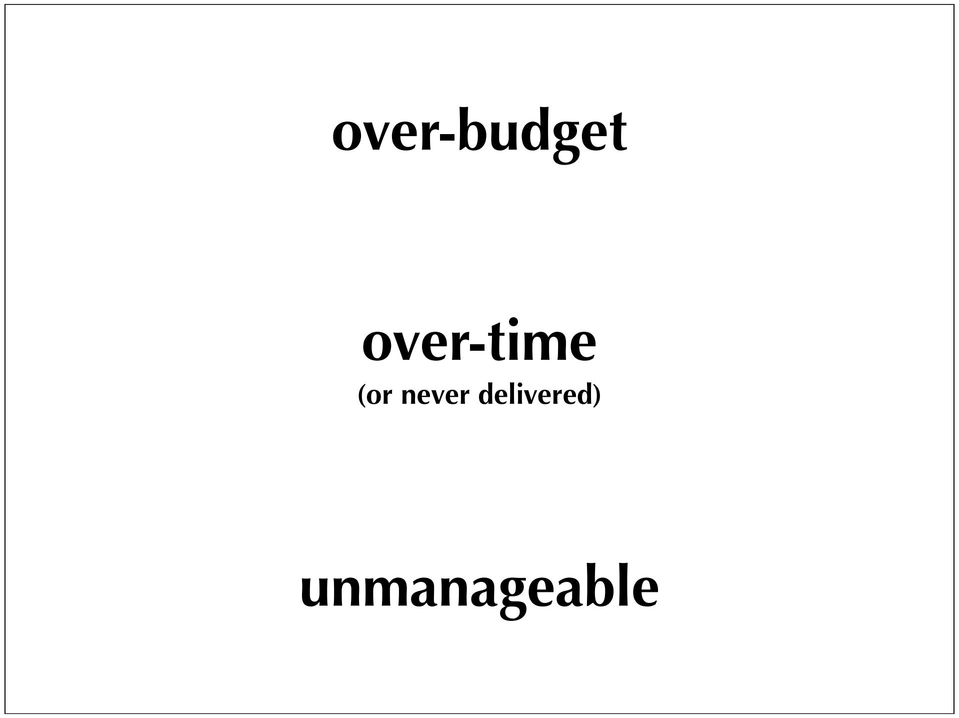
Maintenance accounts for 40-80%
of the cost of software.

Approximately 60% of maintenance
is enhancements, not bug fixes.

More **claims with evidence** also reported by Glass.

(Reminisce about bringing Glass to speak to our undergraduate computer club
using the ACM Distinguished Speakers series.)

The End of the Beginning



over-budget

over-time
(or never delivered)

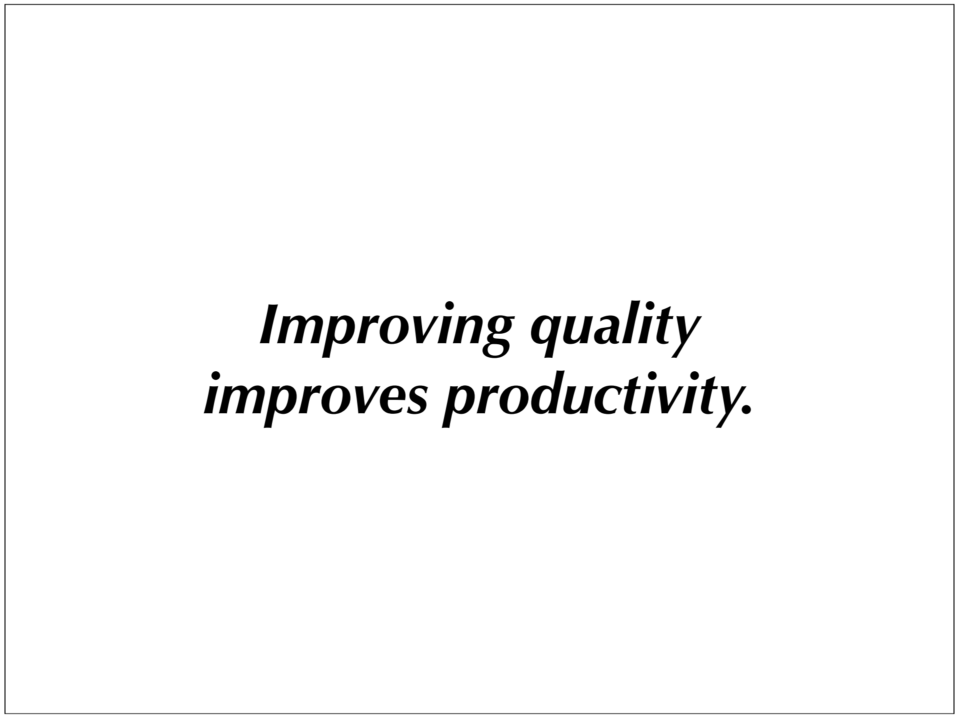
unmanageable

How is this possible?
- low expectations
- lack of accountability
- lack of metrics...

We have solutions.

They aren't evenly distributed.
(One explanation for why good programmers are N times more productive than bad ones.)

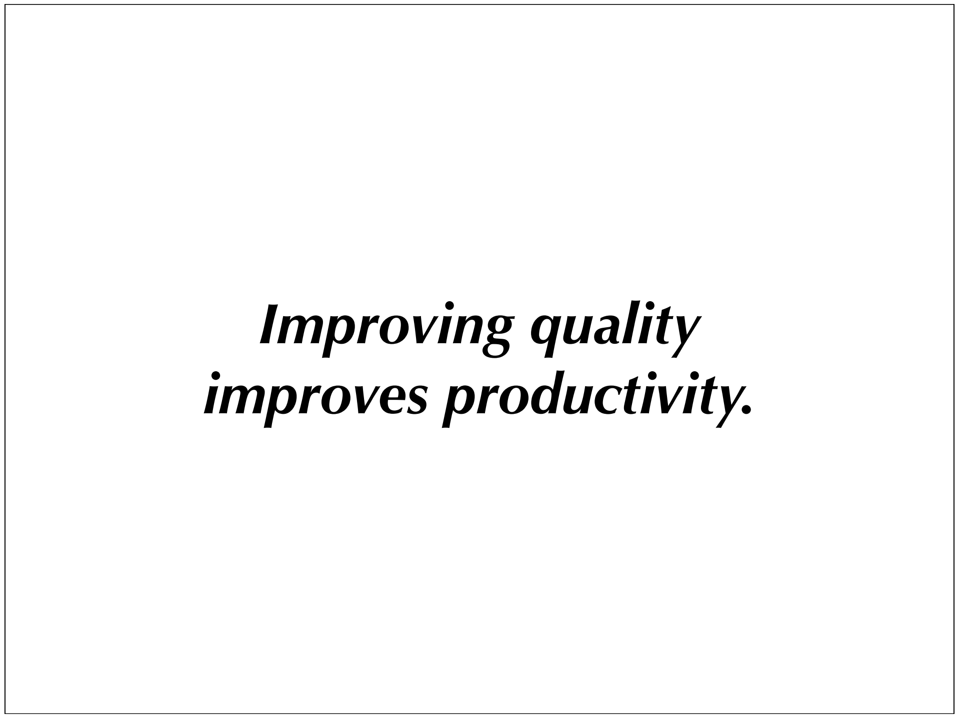
The goal of this course is to introduce you to some of the solutions.



Improving quality improves productivity.

The more effort you put into getting things right, the less time it takes to built it.

The tools and techniques that help you write better code also help you write more code, faster. of the solutions.



Improving quality improves productivity.

The more effort you put into getting things right, the less time it takes to built it.

The tools and techniques that help you write better code also help you write more code, faster. of the solutions.

Acknowledgments

http://software-carpentry.org/lifecycle.html is the source of several of the diagrams and passages used in this session.

Robert L. Glass, **Facts and Fallacies of Software Engineering**, Addison-Wesley Professional, 2002, ISBN 0321117425.