Myths and truths of software development

Chris Cannam, Queen Mary University of London
Greg Wilson, Software Carpentry
Steve Crouch, Software Sustainability Institute

Centre for Intelligent Sensing

Queen Mary University of London

or, what do we *know* about the practice of software development?

(Less than you might hope!)

CIS centre for intelligent sensing

Queen Mary University of London

Computer science

1. Study of how computers work,
   the principles of logic and computation
2. Some largely unscientific ideas
   about how to write programs

[citation needed]

Don’t trust experts, experienced developers, yourself, or me

What does experience count for?

“Programmers were asked several questions about their previous programming experience… number of years of programming experience, total amount of program code written, size of largest program ever written…

“None of these questions had any substantial predictive value for any aspect of programmer performance in the experiment.”

Prechelt, 2003

Theory and evangelism

“That is why *(framework)* is so useful; you can do absolutely anything that you want with it”

“… designed to make programmers happy, and it shows”

“… allows you to do rapid development, and scales to fit”

“I don’t know why one would start a new project in a language that didn’t support *(feature)*”

What passes for evidence

“The best programmers are up to 28 times more productive than the worst”

What passes for evidence

“The best programmers are up to 28 times more productive than the worst”

… in a study comparing batch-processing against interactive systems, using twelve programmers, for one afternoon, over 40 years ago

Sackman, Erikson and Grant, 1968

What *do* we know?

We know some things about working practices

Working together

- Physical distance doesn't matter
- Distance in the organisational structure does

Nagappan et al, 2007; Bird et al, 2009

Working hours & “crunch mode”

8-hour days (a 40-hour week) produce more output than 9-hour days (a 45-hour week)

- **Crunch mode:** going from 40 to 60+ hours a week
  - Initial increase in output
  - Drop-off is obvious within a week
  - By two months, you’d have been better off with 40-hour weeks all along

Abbe, 1908; Robinson, 2005

US Army study

“After circa 24 hours [without sleep] they… no longer knew where they were, relative to friendly and enemy units.

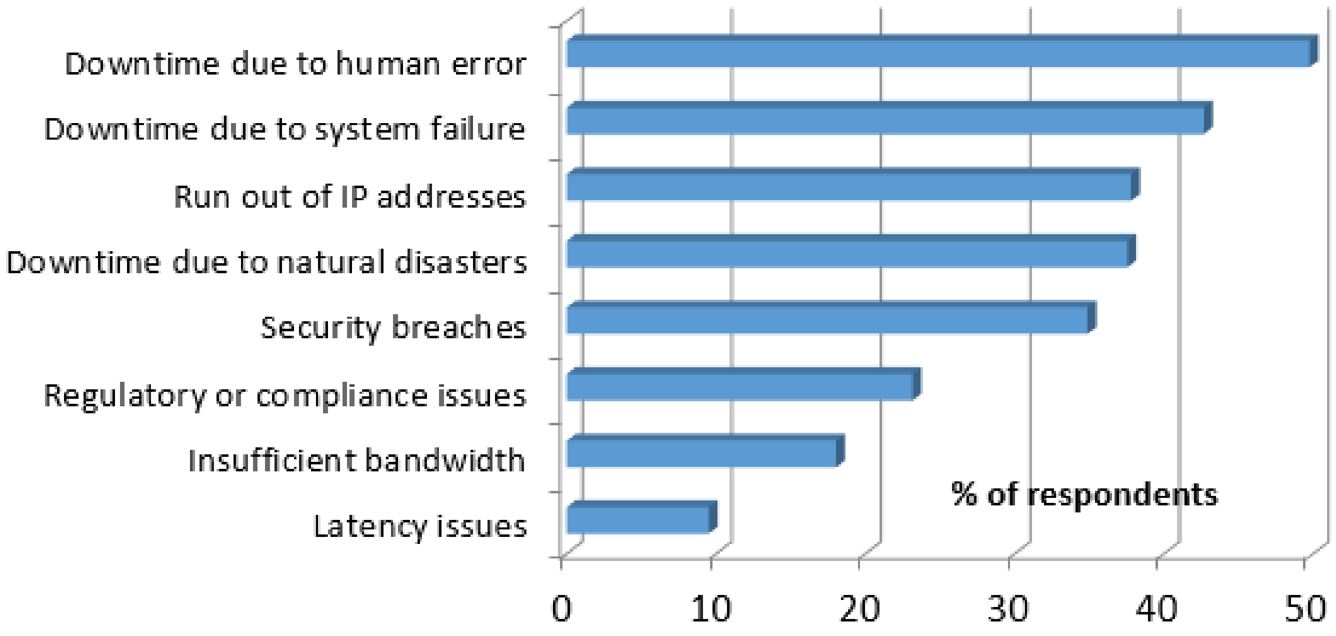
“They no longer knew what they were firing at.

“Early in the simulation, when we called for simulated fire on a hospital, etc., the team would check [and] refuse the request. Later on, they would fire without hesitation regardless of the nature of the target”

Belenky, 1997

Processes and human error

Problems in enterprise datacentres,
as reported to IDC survey



IDC's Enterprise Datacenter Survey, December 2013 (N = 410)

Automate everything but craft

Mistrust yourself in repetitive tasks!

Tools matter

Higher-level languages work

The length of the source code is a predictor of development time, roughly independent of language

Prechelt and Unger, 1999

No current metric is better for defect prediction than lines of code

Herraiz and Hassan, 2010

Variation in run time and memory use due to different programmers is larger than that due to different languages

Prechelt, 2003

Short functions and working memory

Working memory: more or less 7 “things” at once

Weinberg, 1971; Miller, 1956

- The more you have to remember while reading or writing code, the harder it is for you to follow and the less likely to be correct

- Write code for other humans to read – the computer can read anything, it’s humans who matter

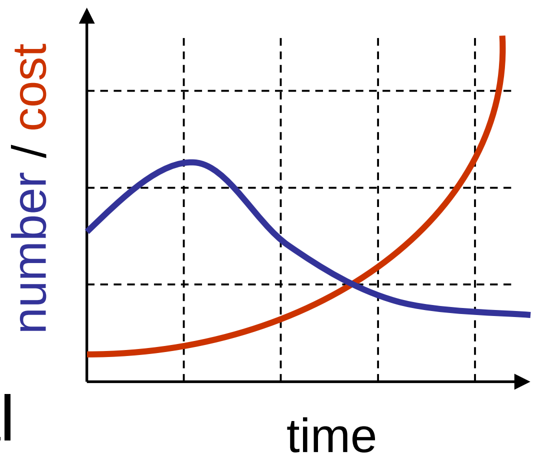
Errors appear early and are costly later

Errors appear early and cost later

Most errors appear during requirements analysis and design

The later an error is detected the more costly it is to address

- 1 hour to fix in the design
- 10 hours to fix in the code
- 100 hours to fix after it's gone live



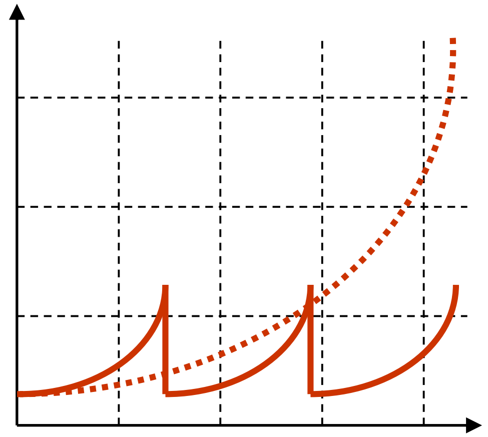
40% of development time is error removal

Boehm et al. 1975; Glass, 1992

Differing approaches

Traditional: "If we take more care at the design stage, fewer bugs will get to the expensive part of the fixing curve"

Agile: "If we do lots of short iterations, the total cost of fixing bugs will go down"



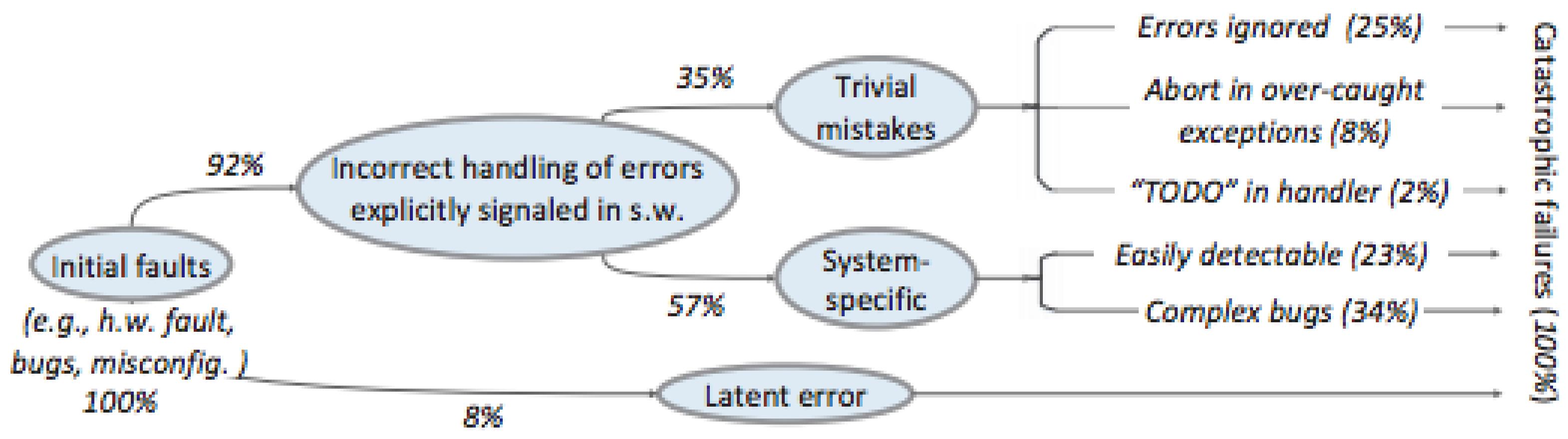
Misunderstandings and misdesigns

30% of errors that survived through to production software were caused by “missing code”, e.g. conditionals where only one branch had been written

(This was the biggest single cause of errors. Regressions were second, at 8.5%)

Glass, 1981

More missing code


Initial faults
(e.g., h.w. fault, bugs, misconfig.)
100%
92%
Incorrect handling of errors explicitly signaled in s.w.
35%
Trivial mistakes
57%
System-specific
8%
Latent error
Errors ignored (25%)
Abort in over-caught exceptions (8%)
"TODO" in handler (2%)
Easily detectable (23%)
Complex bugs (34%)
Catastrophic failures (100%)

92% of catastrophic failures resulted from “incorrect handling of non-fatal errors [that had been] explicitly signalled in software”

Yuan et al, 2014

Bugs are social creatures

Errors tend to cluster:

Half the errors are found in 15% of the modules

Davis, 1995; Endres, 1975

About 80% of the defects come from 20% of the modules, and about half the modules are error free

Boehm and Basili, 2001

When you find errors in one place, watch out for more!

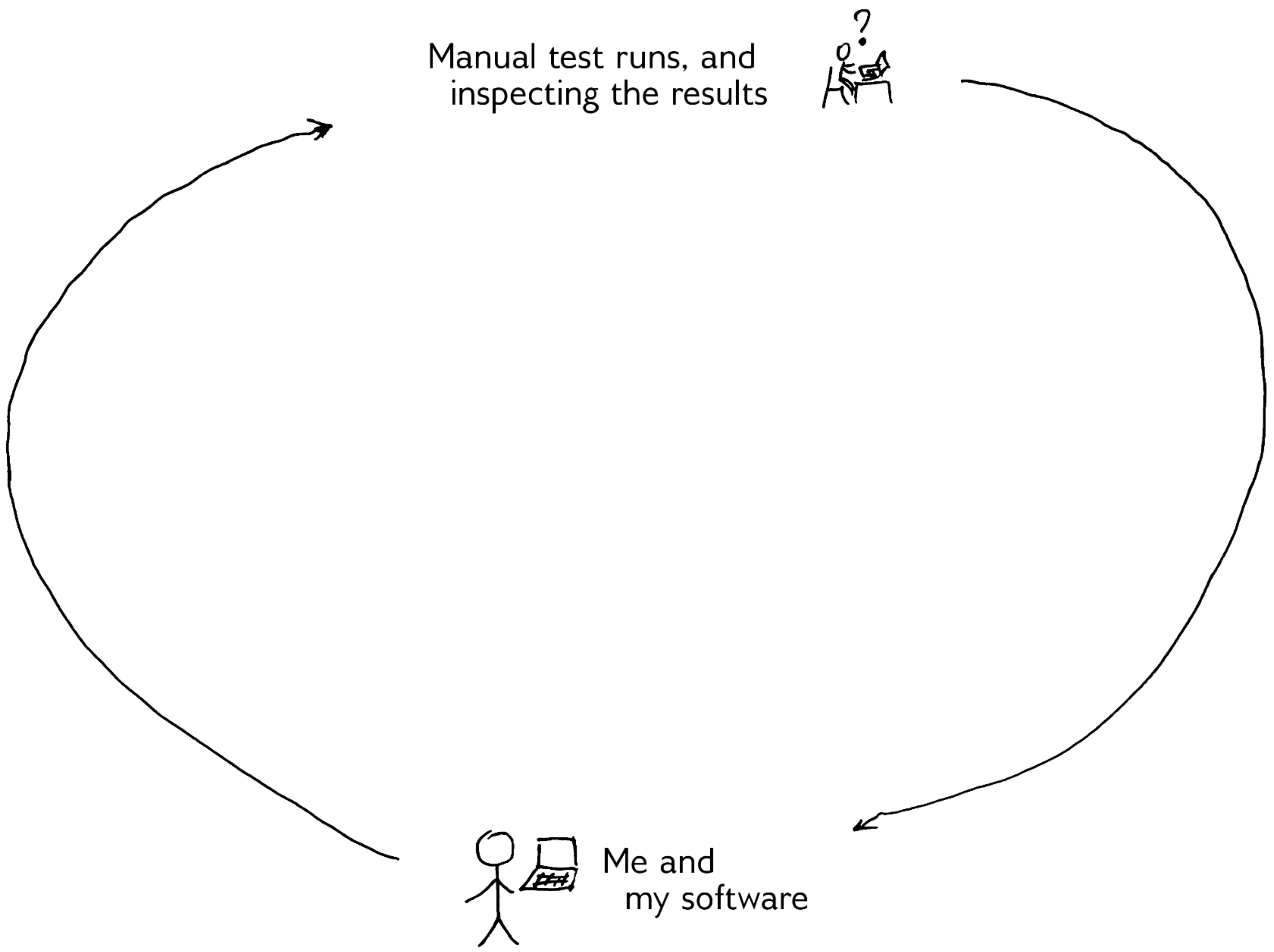
Rapid feedback

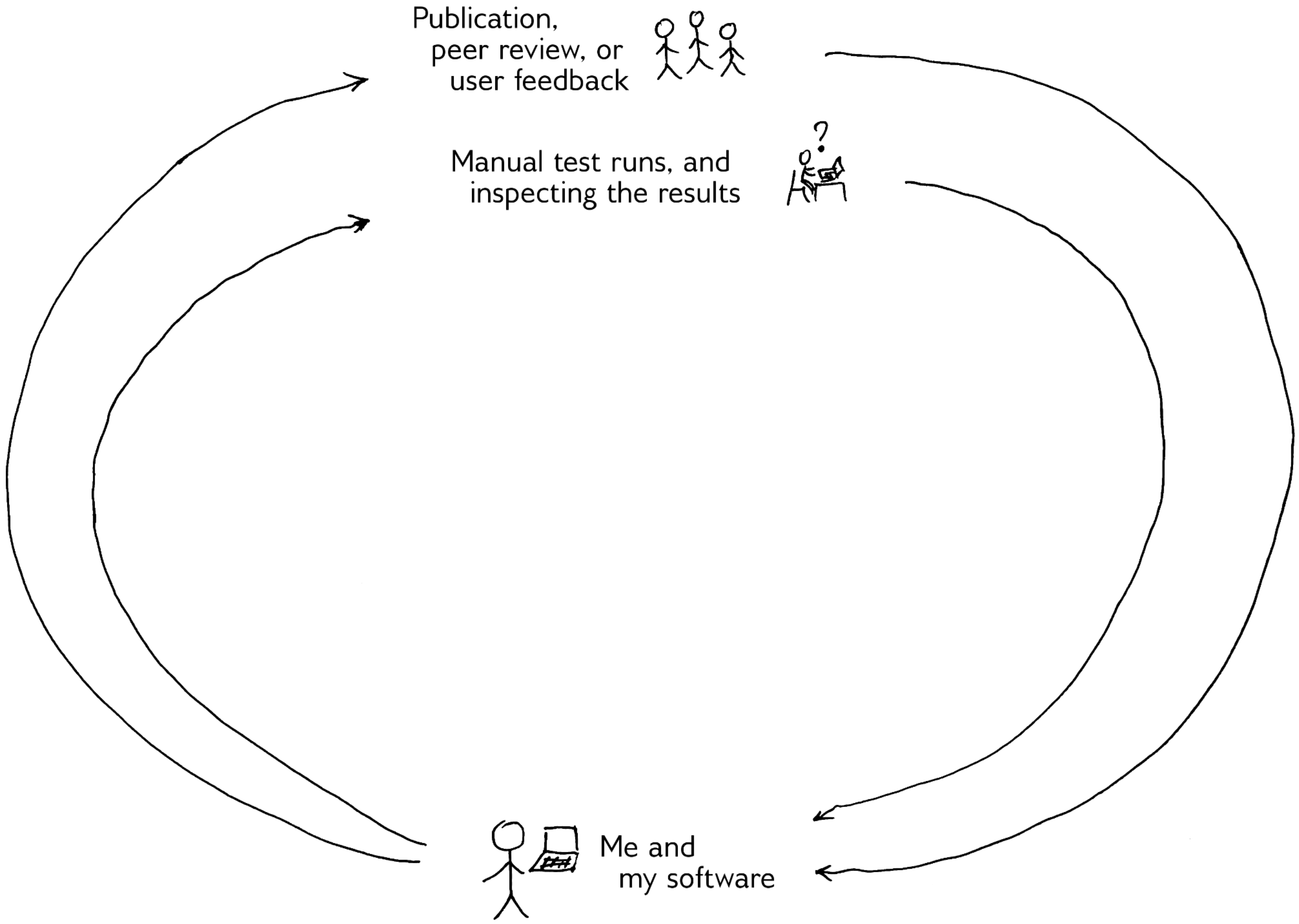
Learn about your mistakes as early as possible

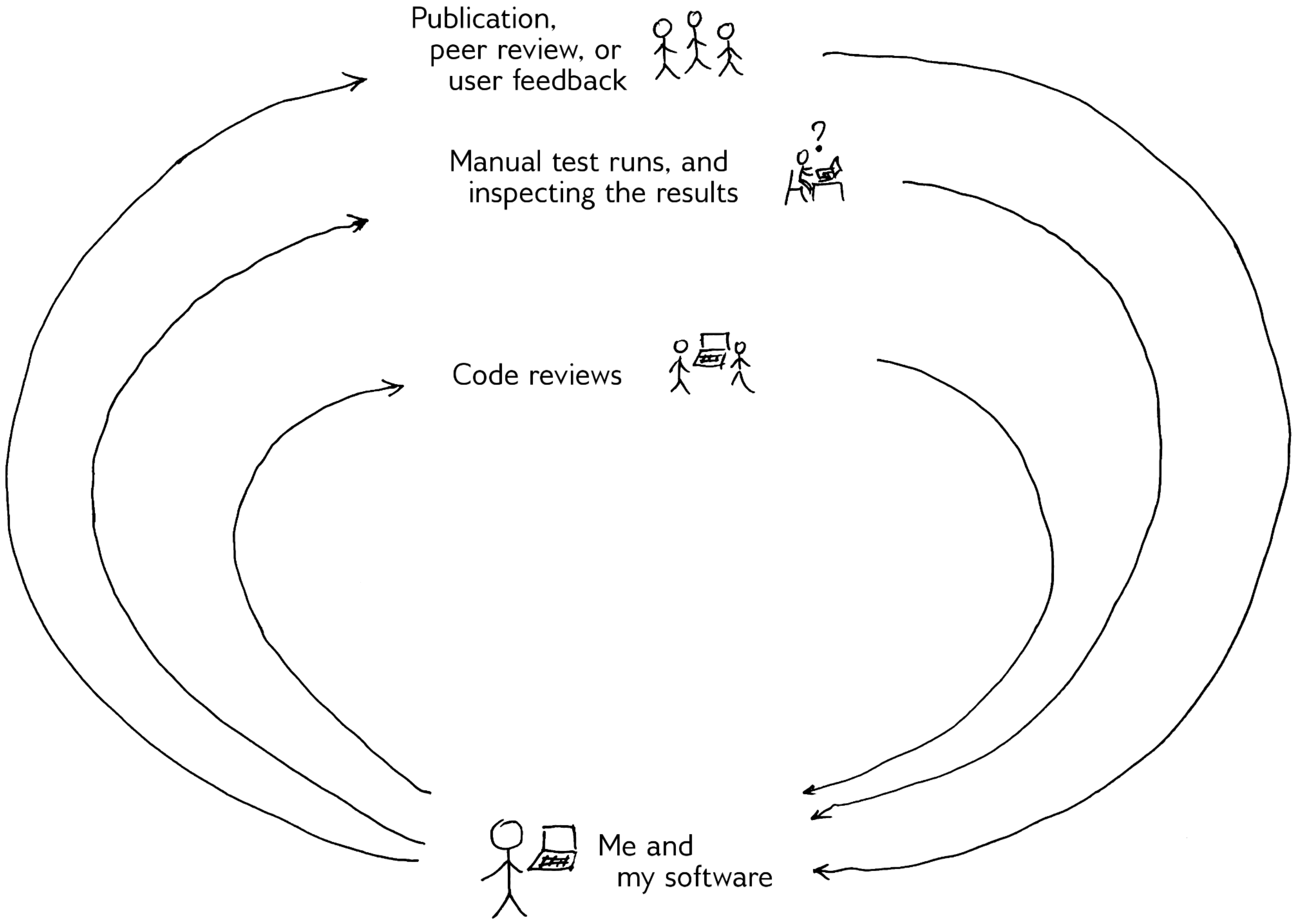
Me and
my software

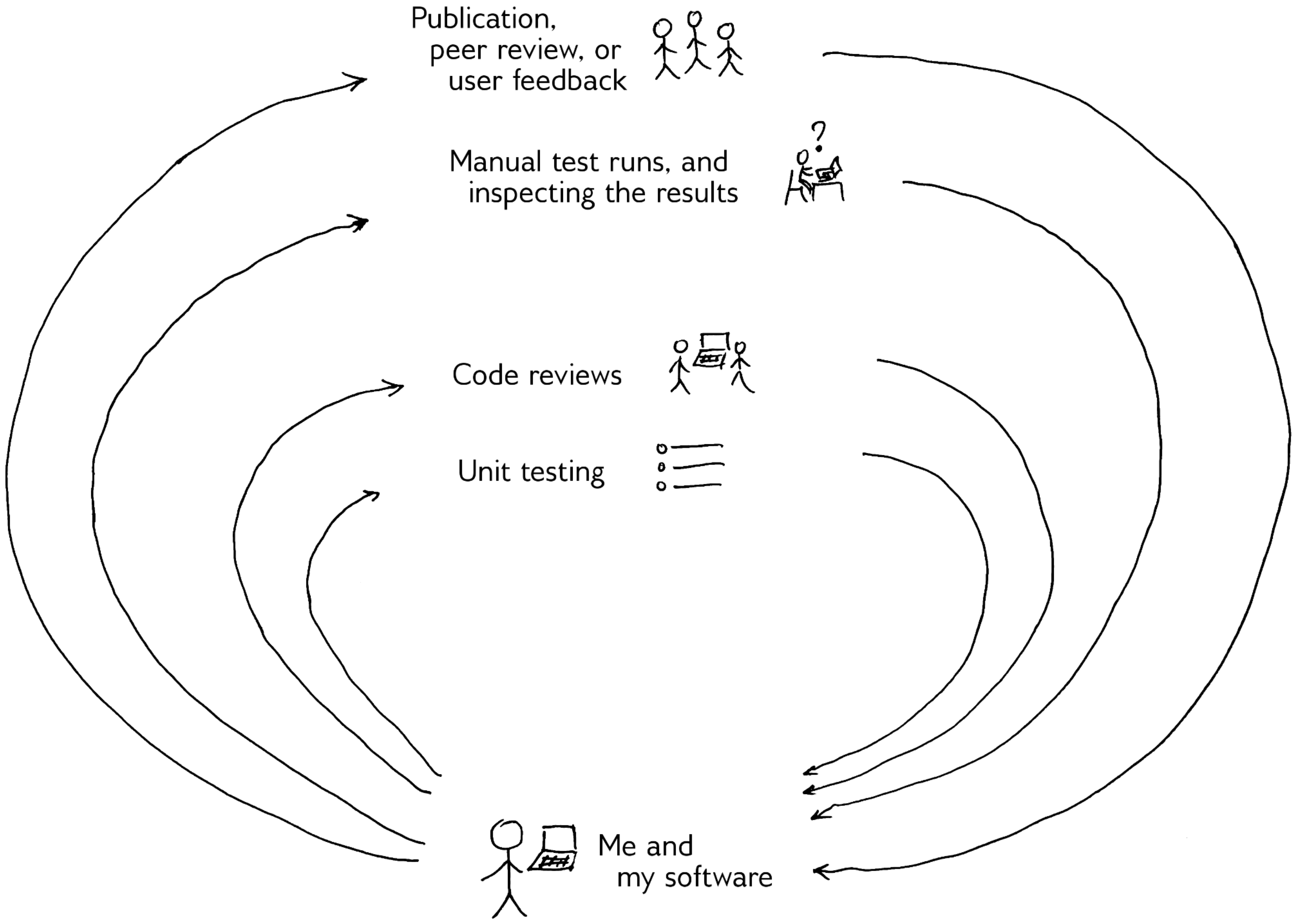
CIS centre for intelligent sensing

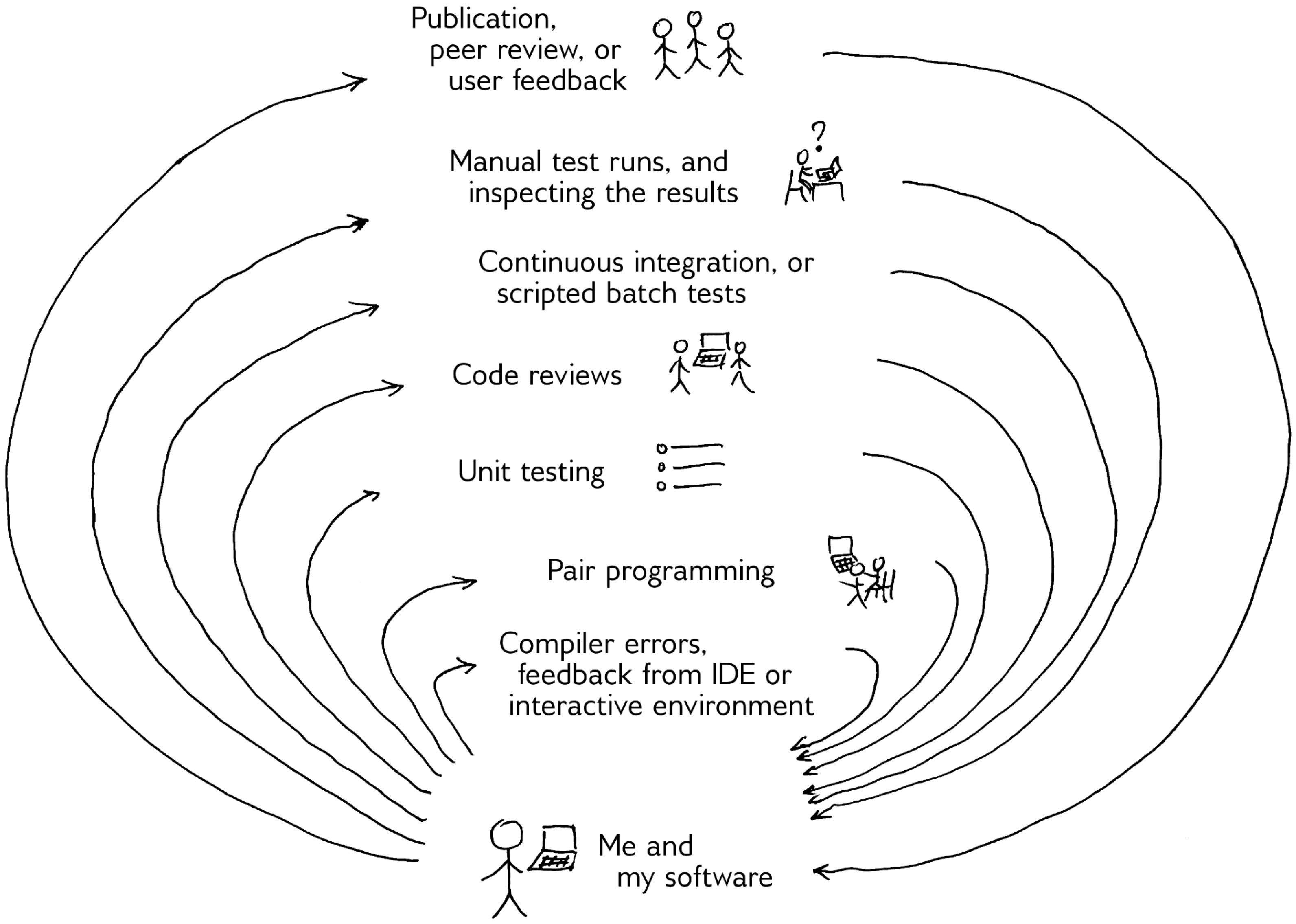
Queen Mary University of London


Manual test runs, and
inspecting the results
Me and
my software


Publication,
peer review, or
user feedback
?
Manual test runs, and
inspecting the results
Me and
my software


Publication,
peer review, or
user feedback
Manual test runs, and
inspecting the results
?
Code reviews
Me and
my software


Publication, peer review, or user feedback
Manual test runs, and inspecting the results
?
Code reviews
Unit testing
Me and my software


Publication, peer review, or user feedback
Manual test runs, and inspecting the results
Continuous integration, or scripted batch tests
Code reviews
Unit testing
Pair programming
Compiler errors, feedback from IDE or interactive environment
Me and my software

Read and share

Reading and code reviews

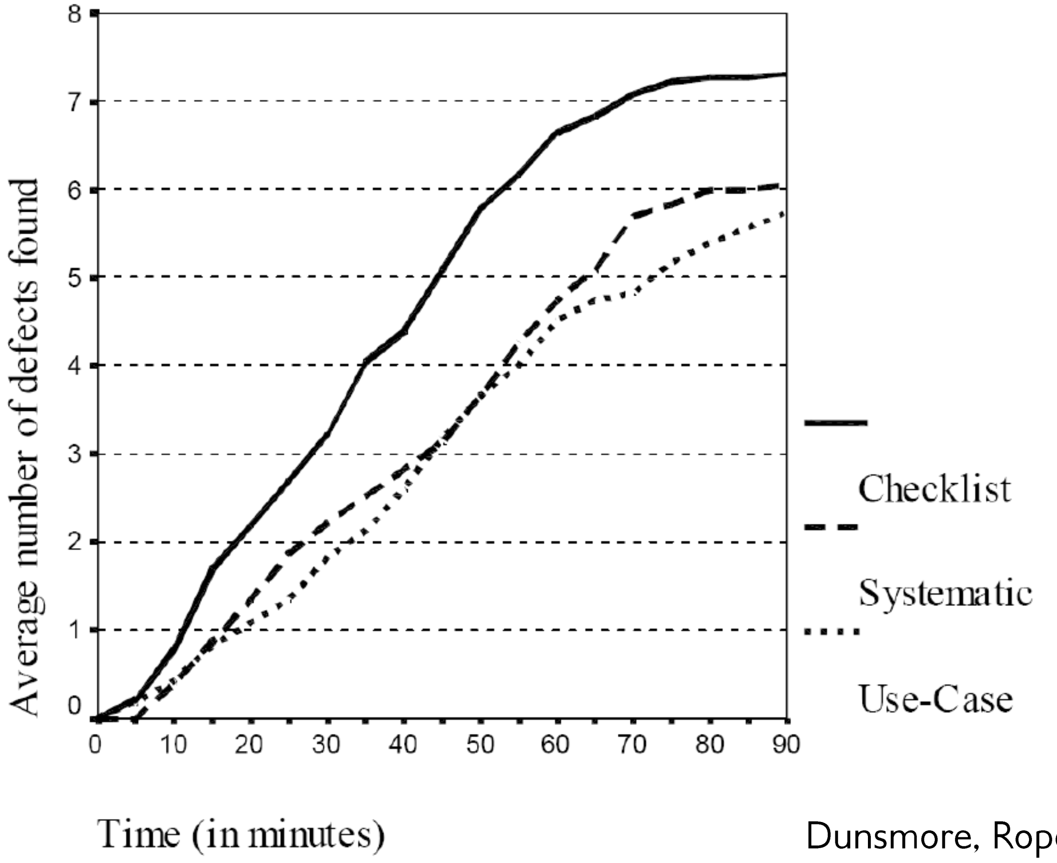
Inspections can remove 60-90% of errors before the first test is run

Fagan, 1975

Training developers in “how to read code” makes a substantial difference to the number of errors found during code reviews (compared with more formal techniques)

Rifkin and Deimel, 1995

Code reviews: not so demanding



Dunsmore, Roper, Wood, 2000.
From *Best-Kept Secrets of Peer Code Review*, Cohen, 2006

Review of “obvious” problems

“In 23% of the catastrophic failures, the error handling logic of a non-fatal error was so wrong that any statement coverage testing or more careful code reviews by the developers would have caught the bugs”

Yuan et al, 2014

We often don’t like to share

- Survey of papers in economics journal **with** a data and code archive policy:

- 9 empirical articles

McCullough, 2007

We often don’t like to share

- Survey of papers in economics journal **with** a data and code archive policy:

- 9 empirical articles
  - 7 had empty entries in the journal archive!

McCullough, 2007

We often don’t like to share

- Survey of papers in economics journal **with** a data and code archive policy:

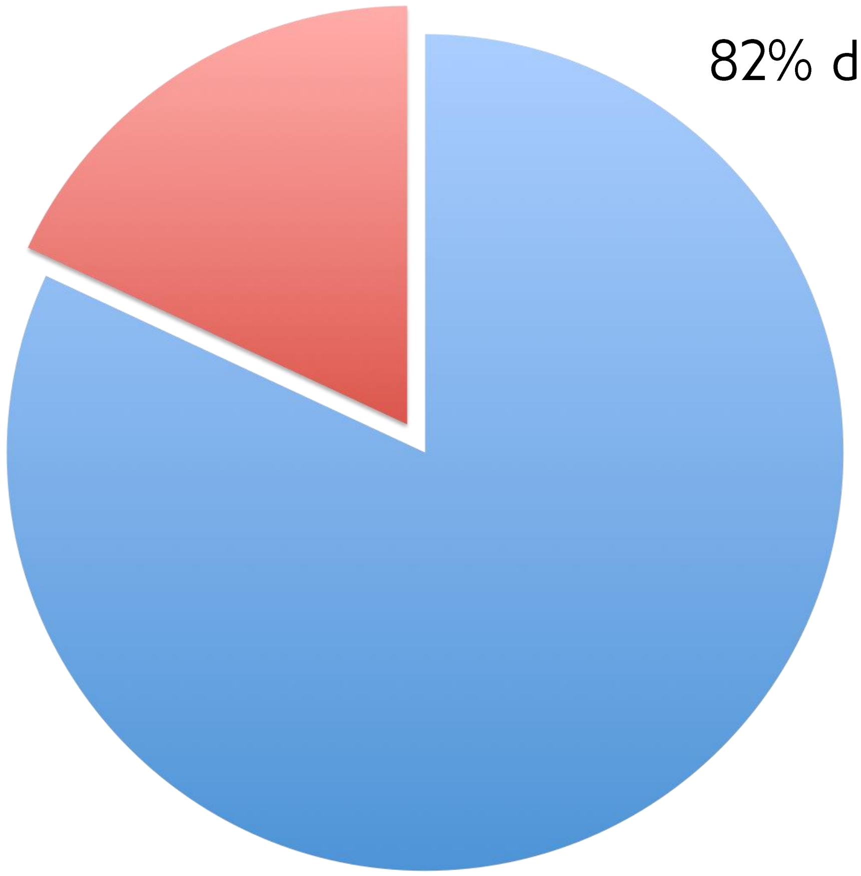
- 9 empirical articles
  - 7 had empty entries in the journal archive!
  - The other two had code, but it didn't work!
  - None of them could be replicated without authors' help

McCullough, 2007

Hmmm

J M Wicherts, M Bakker and D Molenaar, *Willingness to Share Research Data Is Related to the Strength of the Evidence and the Quality of Reporting of Statistical Results*, 2011

SoundSoftware survey 2010–2011

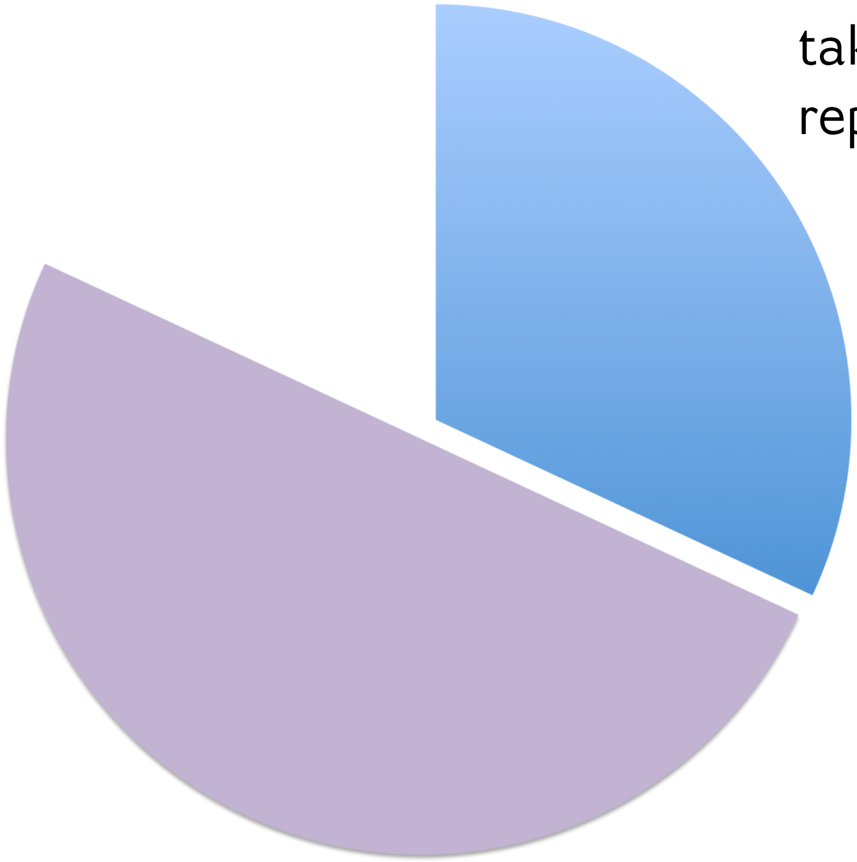

82% develop code

CIS centre for intelligent sensing

Queen Mary University of London

SoundSoftware survey 2010–2011

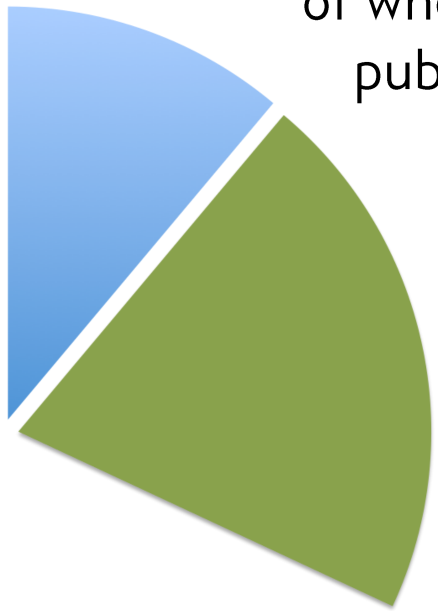
of whom 39% report taking steps to reproducibility



CIS centre for intelligent sensing

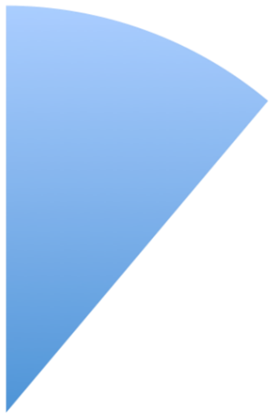
Queen Mary University of London

SoundSoftware survey 2010–2011



of whom 35% report publishing any code

SoundSoftware survey 2010–2011



That's 11% of the whole

CIS centre for intelligent sensing

Queen Mary University of London

Learn to read!

You don't just go out and write War and Peace

- You'd read other skilled writers first

You don't just go out and write a foreign language

- You learn to read it first

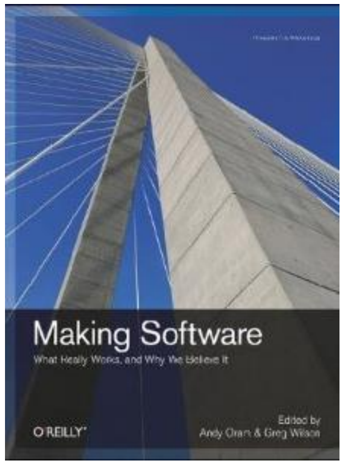
…and prepare to share

Coding with readers in mind makes it

- easier to write comments you'll understand later
- easier to write testable code, and to write tests for it
- easier to do code reviews
- and easier to contemplate publishing

Further reading

Facts and Fallacies of Software Engineering
Robert L. Glass
Foreword by Alan M. Davis


Making Software
What Really Works, and Why We Believe It
O'REILLY
Edited by Andy Oram & Greg Wilson

"Best Practices for Scientific Computing" – Wilson et al. *http://arxiv.org/abs/1210.0530*

"Facts and Fallacies of Software Engineering" – Robert L Glass

"Making Software" – edited by Oram and Wilson

See also *http://software-carpentry.org*

And finally

Get into good habits early!

It’s too easy to cling to bad ones