

Pointers to more effective software & data in audio research

Mark D Plumbley, Chris Cannam, Steve Welburn
Centre for Digital Music
Queen Mary, University of London

Research software: The Question

“How do you know your code implements your method at all?”

Validation and verification

Validation: Establishing how well your model represents reality

- Research papers are expected to do this

Validation and verification

Validation: Establishing how well your model represents reality

- Research papers are expected to do this

Verification: Establishing that you have implemented your model

- This is what formal software testing is about
- It's also what most informal good practice aims at
- Research papers seldom attempt any of this

“Feedback cycles”

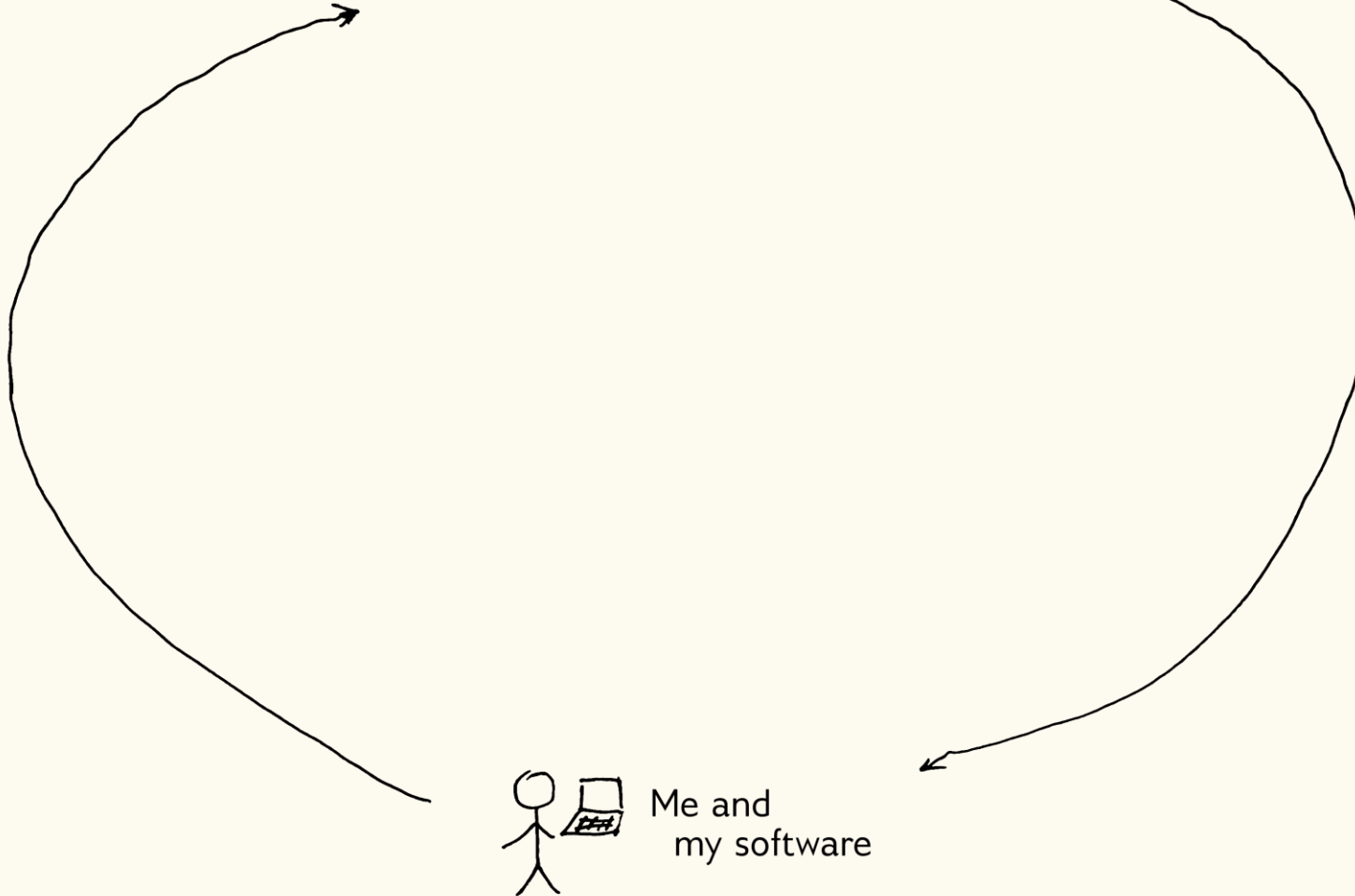
Write code → learn how code is wrong → fix code

- Software development “best practice” is often about trying to shorten or simplify this cycle
- Learn about your mistakes as early as possible



Me and
my software

Manual test runs, and
inspecting the results



Me and
my software

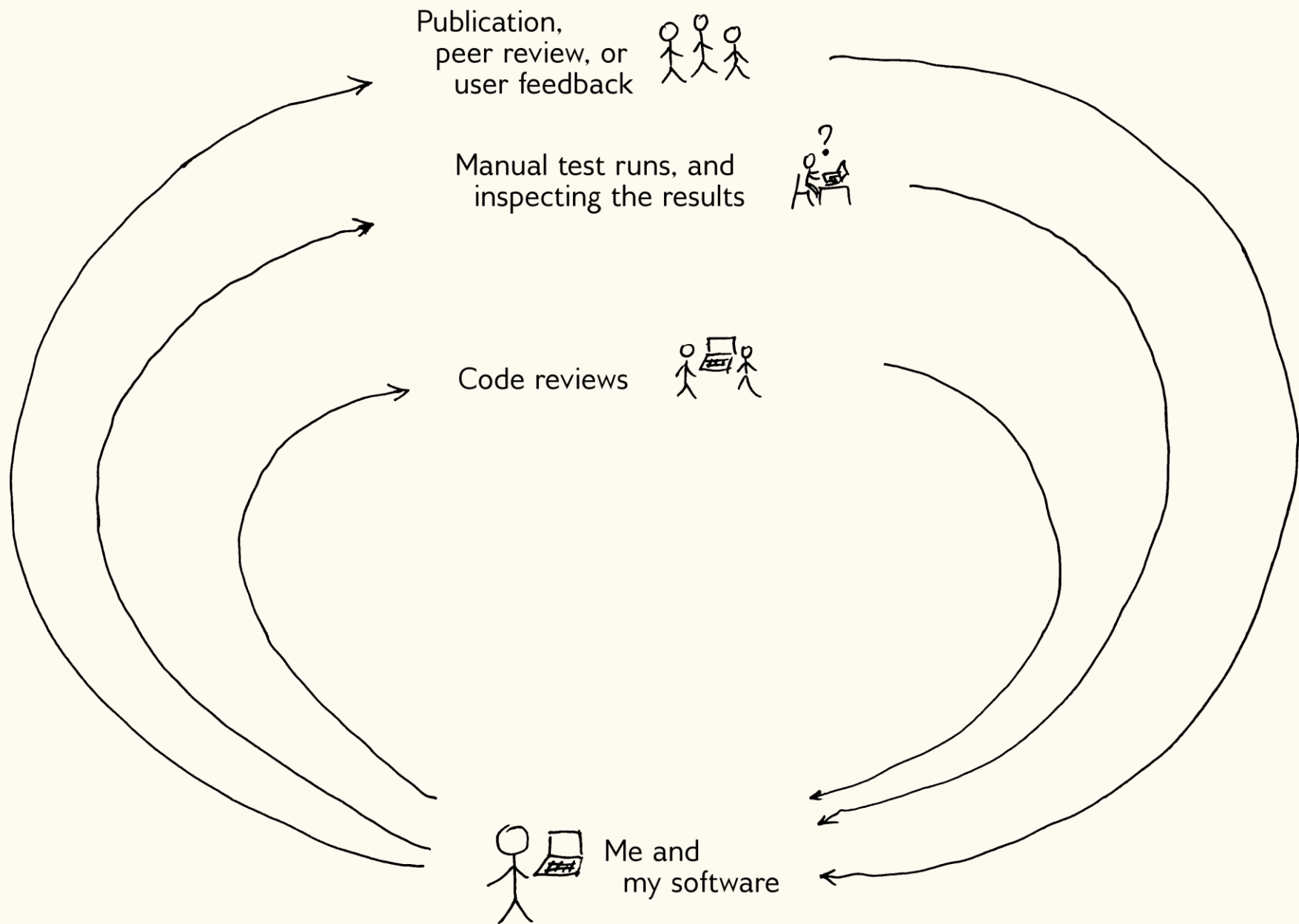
Publication,
peer review, or
user feedback

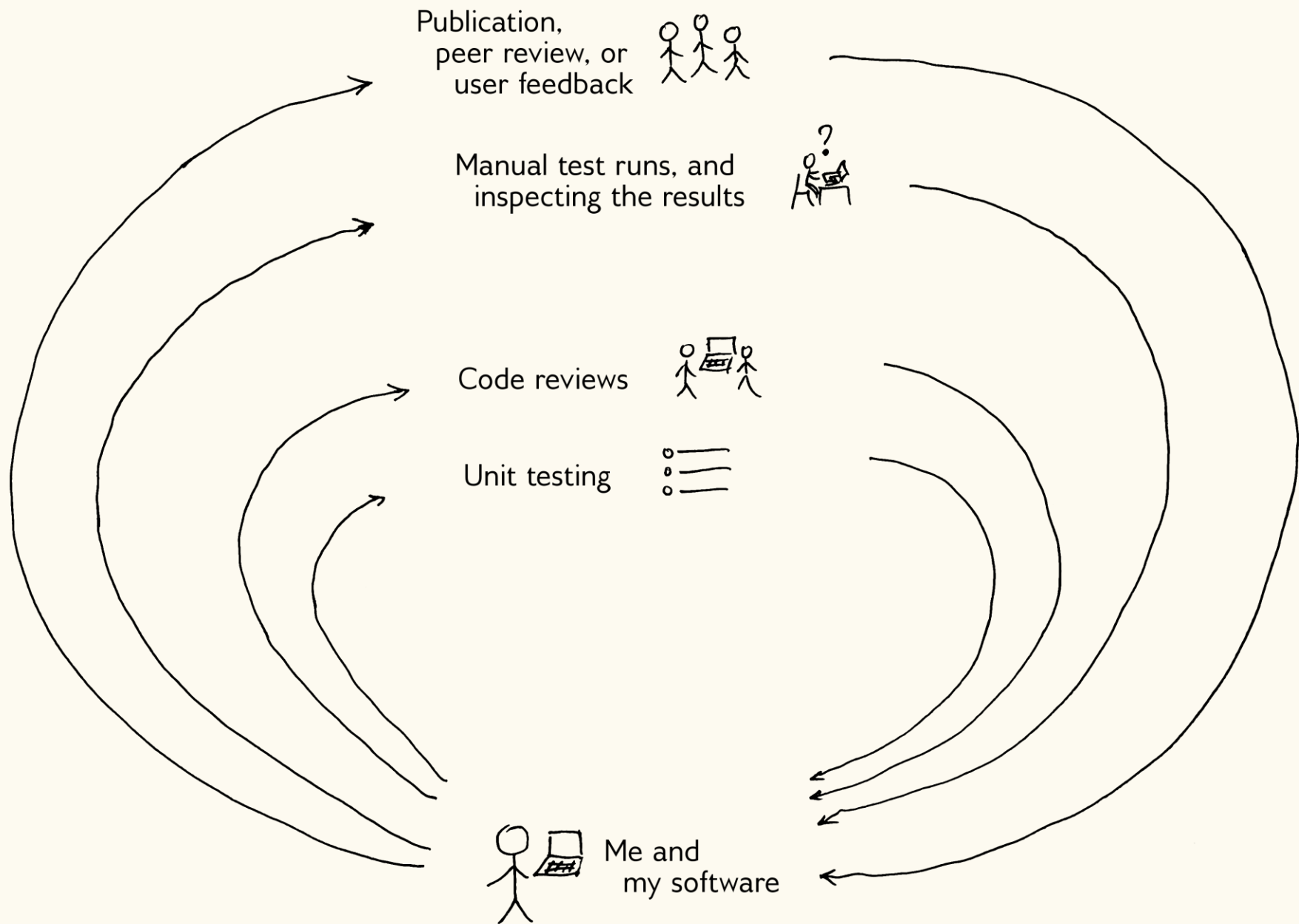


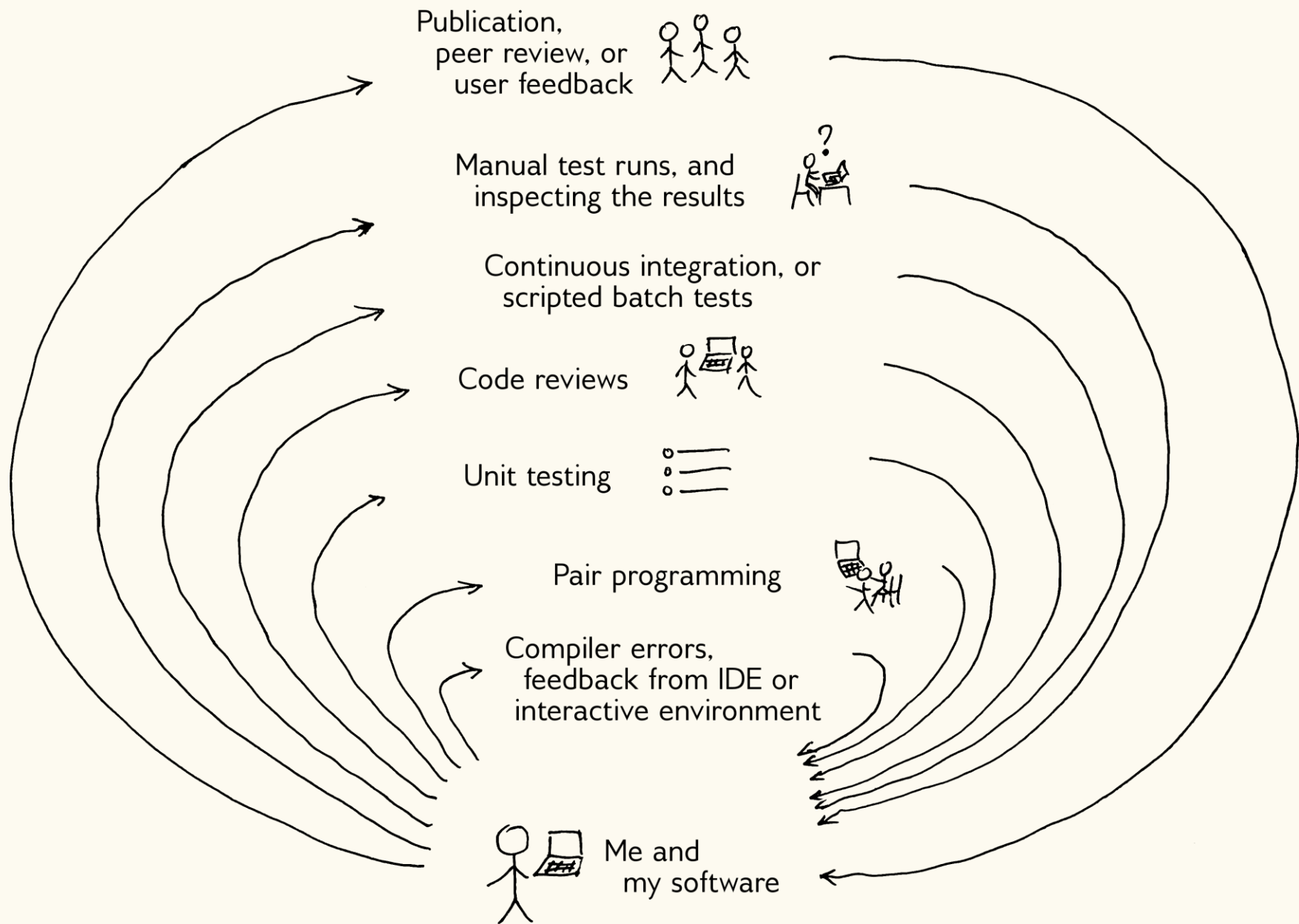
Manual test runs, and
inspecting the results



Me and
my software







A philosophical point

Much of this involves becoming more comfortable with the idea that *someone else will be looking at your code*.

(We collaborate on papers; why not so much on code?)

Coding with readers in mind makes some things easier:

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

“Code reviews”

Simply: getting someone else to read your code!

Can be informal, in a peer setting

- use-case reviewing: following the flow from known inputs

Or a bit more formal

- checklist reviewing: looking for likely troublespots

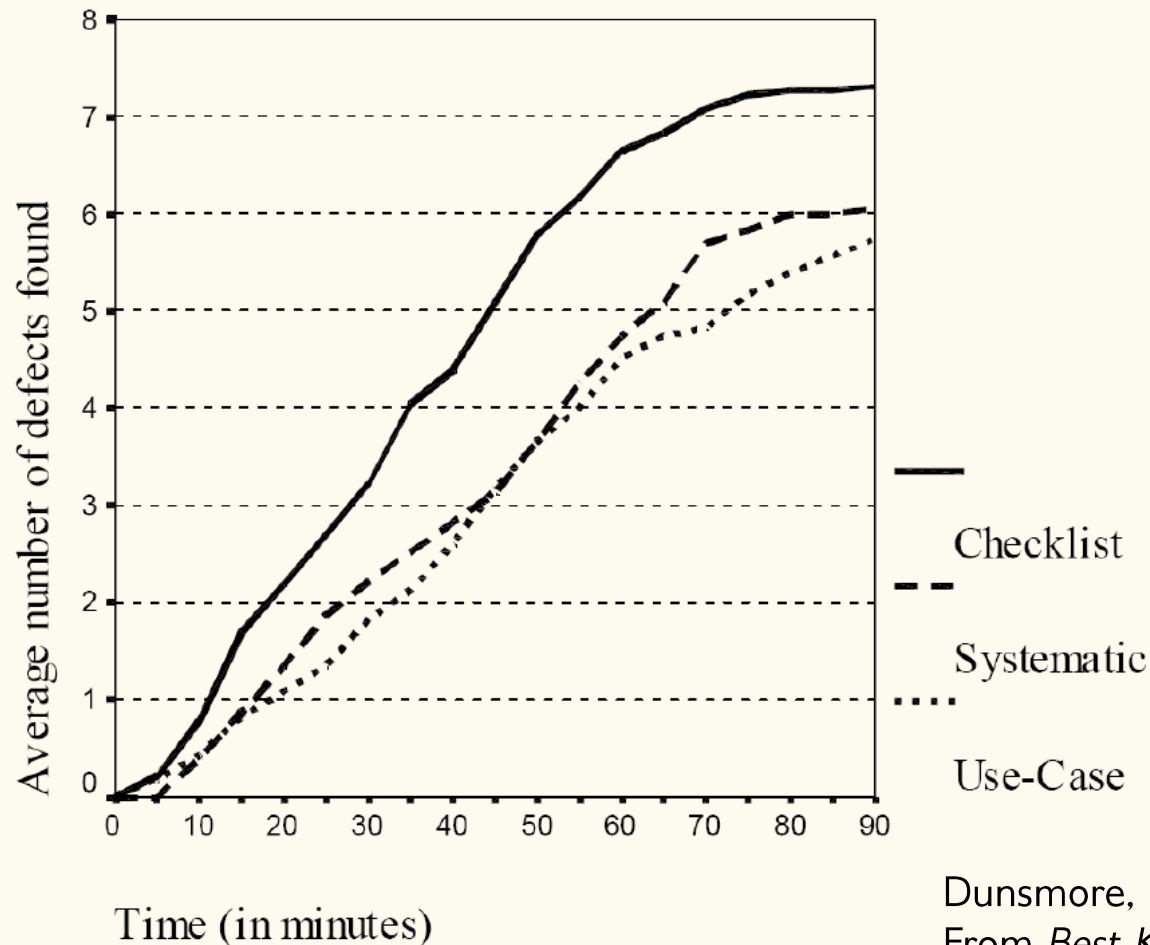
Can move from one to the other through simple application of experience!

Readable code is reviewable code

- readable variable names
- slicing into methods
- commenting the why, not the how
- concentrate on function prototypes / API description
- check the SWC paper?

Double-check what Cohen suggested as a review checklist

Code reviews? No problem



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

Unit testing: what is it?

A “unit test” is a bit of code that calls one of your functions, gives it some input, and tells you whether it returned the right result

Write a set of these, and you have a “test suite”

A “test framework” can help you write them more quickly; there's at least one for every programming language and environment

Should be set up so you can run all tests in one go

Unit testing: what's it for?

An automated way of ensuring:

- That your code's API works
- That the individual parts of your code work correctly
- That you don't break your code when changing it

Also useful when developing a tricky algorithm (using test-first, or test-driven development)

Unit testing questions

“How do I write tests when I don’t know what results to expect?”

Unit testing questions

“How do I write tests when I don’t know what results to expect?”

- Break it down into functions whose behaviour you can predict
- Testable code is also more readable code
- Test individual components, not the whole thing

Unit testing is about trying to ensure that *the code implements the method*—not that *the method is the right one*

Unit testing questions

“How do I write tests when I don’t know what results to expect?”

“...if we’ve learned anything from the agile movement in the last 15 years, it’s that the more improvisatory your development process is, the more important careful craftsmanship is as well—unless, of course, you don’t care whether your programs are producing correct answers or not.”

<http://software-carpentry.org/2012/09/not-really-disjoint/>

Unit testing questions

“What sort of test data and test cases should I write?”

- The simplest possible ones!

“But I have big data sets and complex results!”

- Don't use real-world data: that's a different kind of test
- Look for the smallest possible input to test a given behaviour

An example

Audio-to-note system for solo vocal music recordings:

<http://code.soundsoftware.ac.uk/projects/cepstral-pitchtracker>

- Overlapping time-domain windower
- Short-time Fourier transform using FFT library
- Transform to cepstrum
- Peak interpolator
- Multi-agent method for identifying candidate notes
- Estimator for perceived pitch from peak frequency series

An example

We don't know what output we expect for real-world data, but every one of its components can be tested

- Overlapping time-domain windower ✓
- Short-time Fourier transform using FFT library ✓
- Transform to cepstrum ✓
- Peak interpolator ✓
- Multi-agent method for identifying candidate notes ✓
- Estimator for perceived pitch from peak frequency series ✓

Version control

Software to help keep track of changes made to files

- Tracks the **history** of your work
- Helps you **collaborate** with others

Popular examples include git, Mercurial, Subversion

Keeping track of history

- How do you get back to that working version you had yesterday?
- How do you get from “it’s not working!” to understanding what went wrong?
- How would you repeat the experiments from that journal paper you wrote last year?

Collaborating... with yourself!

You need to run the same software on your laptop at home and the server in the lab:

- How do you get the code onto both?
- How do you *verify* that you have the same code on both?

Collaborating with others

You're working on code or a paper with a colleague...

- How do you find out when they change something?
- How do you merge your changes without getting in a mess?
- How can you find out which of you introduced a bug, and when?

Version control helps with all this

A version control system

- Records your files' history for you
- Shows differences between different versions
- Handles sync between copies on different computers

But you do have to work a bit:

- Tell it which files are part of your project
- Tell it when you've changed something
- If two people make *conflicting* changes, one of them must *resolve* them

Version control systems

Subversion: Centralised

- one server, one repository database
- every commit goes straight to the shared repository

Mercurial, git: Distributed

- every working copy has complete repository in it
- commits are local, then push/pull between computers
- e.g. work locally, then push to a remote hosting site

Version control hosting sites

General-purpose :

- GitHub: very popular for sharing Git repositories
- BitBucket: Git and Mercurial, good private repo support
- SourceForge: the old-school option for open source projects

Thematic:

- `code.soundsoftware.ac.uk`: for the UK audio and music research community

Very specific:

- Does your department provide hosting?

How is that different from Dropbox?

Consistency guarantees:

- Software needs to be *exactly as written* across all files
- Files changed together must be updated together
- Changes to a file by different people must be merged correctly

Record keeping:

- Publish and replicate history reliably
- Interrogate past changes and find out what version you are looking at

Typical version control questions

- When should I start using version control for my project?
- Which files should I track in the repository?
- How often should I commit?
- How often should I push changes to a shared repo?

Five things to do tomorrow

- Get your current research code into a version control repository, push to a hosting site (a private project is fine)
- Pull it onto another computer, get it to build and run
- Talk to another researcher in your group, allot an hour to experiment with reviewing a bit of each other's code
- Identify a piece of your code that you can isolate and test
- Pick a licence for your code, check compatibility with other code you're using, add it to code files